Oracle® Machine Learning for SQL

API Guide

23ai
F47584-04
May 2024

Oracle Machine Learning for SQL API Guide, 23ai

F47584-04

Primary Author: Sarika Surampudi

Contents

Part II Machine Learning Techniques

3 Regression

4 Classification

Part III Algorithms

14 Apriori

15 CUR Matrix Decomposition

16 Decision Tree

17 Expectation Maximization

18 Explicit Semantic Analysis

19 Exponential Smoothing

27 O-Cluster

28 R Extensibility

29 Random Forest

30 Singular Value Decomposition

31 Support Vector Machine

36 Create a Model

37 Scoring and Deployment

38 Machine Learning Operations on Unstructured Text

39 Integration of ONNX Runtime

40 Administrative Tasks for Oracle Machine Learning for SQL

Preface

This preface contains the following topics:

Technology Rebrand

Oracle is rebranding the suite of products and components that support machine learning with Oracle Database and Big Data. This technology is now known as Oracle Machine Learning (OML).

The OML application programming interfaces (APIs) for SQL include PL/SQL packages, SQL functions, and data dictionary views. Using these APIs is described in publications, previously under the name Oracle Data Mining, that are now named Oracle Machine Learning for SQL (OML4SQL).

Audience

This guide is intended for application developers and database administrators who are familiar with SQL programming and Oracle Database administration and who have a basic understanding of machine learning concepts.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

For more information, see these Oracle resources:

- Oracle Public Cloud

  `http://cloud.oracle.com`
- *Oracle Machine Learning for SQL Concepts*
- *Oracle Machine Learning for SQL User's Guide*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Database Reference*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I
Introductions

Part I presents an introduction to Oracle Machine Learning for SQL. The first chapter is a general, high-level overview for those who are new to machine learning technology.

Part I contains the following chapters:

- Introduction to Oracle Machine Learning for SQL
- Oracle Machine Learning Basics

1 Introduction to Oracle Machine Learning for SQL

Introduces Oracle Machine Learning for SQL to perform a variety of machine learning tasks.

- About Oracle Machine Learning for SQL
- Oracle Machine Learning for SQL in the Database Kernel
- Oracle Machine Learning for SQL with R Extensibility
- Oracle Machine Learning for SQL in Oracle Exadata
- About Partitioned Models
- Interfaces to Oracle Machine Learning for SQL
- Overview of Database Analytics

1.1 About Oracle Machine Learning for SQL

Oracle Machine Learning for SQL (OML4SQL) provides scalable in-database machine learning algorithms through PL/SQL and SQL APIs. The algorithms are fast and scalable, support algorithm-specific automatic data preparation, and can score in batch or real-time.

OML4SQL provides a powerful, state-of-the-art machine learning capability within Oracle Database. The parallelized algorithms in the database keep data under database control. There is no need to extract data to separate machine learning engines, which adds latency to data access and raises concerns about data security, storage, and recency. The algorithms are fast and scalable, support algorithm-specific automatic data preparation, and can score in batch or real-time. You can use OML4SQL to build and deploy predictive and descriptive machine learning applications, to add intelligent capabilities to existing applications, and to generate predictive queries for data exploration. OML4SQL provides explanatory prediction details when scoring data, so you can understand why an individual prediction is made.

OML4SQL offers a broad set of in-database algorithms for performing a variety of machine learning tasks, such as classification, regression, anomaly detection, feature extraction, clustering, and market basket analysis. The algorithms can work on standard case data, transactional data, star schemas, and unstructured text data. OML4SQL is uniquely suited to the analysis of very large data sets.

Oracle Machine Learning for SQL, along with Oracle Machine Learning for R and Oracle Machine Learning for Python, is a component of Oracle Machine Learning that provides three powerful APIs for in-database machine learning, among other features.

1.2 Oracle Machine Learning for SQL in the Database Kernel

Learn about the implementation of Oracle Machine Learning for SQL (OML4SQL) in Oracle Database kernel and its advantages.

OML4SQL is implemented in the Oracle Database kernel. OML4SQL models are first class database objects. Oracle Machine Learning for SQL processes use built-in features of Oracle Database to maximize scalability and make efficient use of system resources.

OML4SQL within Oracle Database offers many advantages:

- No Data Movement: Some machine learning products require that the data be exported from a corporate database and converted to a specialized format. With OML4SQL, no data movement or conversion is needed. This makes the entire process less complex, time-consuming, and error-prone, and it allows for the analysis of very large data sets.
- Security: Your data is protected by the extensive security mechanisms of Oracle Database. Moreover, specific database privileges are needed for different machine learning activities. Only users with the appropriate privileges can define, manipulate, or apply machine learning model objects.
- Data Preparation and Administration: Most data must be cleansed, filtered, normalized, sampled, and transformed in various ways before it can be mined. Up to 80% of the effort in a machine learning project is often devoted to data preparation. OML4SQL can automatically manage key steps in the data preparation process. Additionally, Oracle Database provides extensive administrative tools for preparing and managing data.
- Ease of Data Refresh: Machine learning processes within Oracle Database have ready access to refreshed data. OML4SQL can easily deliver machine learning results based on current data, thereby maximizing its timeliness and relevance.
- Oracle Database Analytics: Oracle Database offers many features for advanced analytics and business intelligence. You can easily integrate machine learning with other analytical features of the database, such as statistical analysis and analytic views.
- Oracle Technology Stack: You can take advantage of all aspects of Oracle's technology stack to integrate machine learning within a larger framework for business intelligence or scientific inquiry.
- Domain Environment: Machine learning models have to be built, tested, validated, managed, and deployed in their appropriate application domain environments. Machine learning results may need to be post-processed as part of domain specific computations (for example, calculating estimated risks and response probabilities) and then stored into permanent repositories or data warehouses. With OML4SQL, the pre- and post-machine learning activities can all be accomplished within the same environment.
- Application Programming Interfaces: The PL/SQL API and SQL language operators provide direct access to OML4SQL functionality in Oracle Database.

Related Topics

- Overview of Database Analytics

1.3 Oracle Machine Learning for SQL in Oracle Exadata

Understand how complex scoring and algorithmic processing is done using Oracle Exadata.

Scoring refers to the process of applying a OML4SQL model to data to generate predictions. The scoring process may require significant system resources. Vast amounts of data may be involved, and algorithmic processing may be very complex.

With OML4SQL, scoring can be off-loaded to intelligent Oracle Exadata Storage Servers where processing is extremely performant.

Oracle Exadata Storage Servers combine Oracle's smart storage software and Oracle's industry-standard hardware to deliver the industry's highest database storage performance. For more information about Oracle Exadata, visit the Oracle Technology Network.

Related Topics

- https://www.oracle.com/engineered-systems/exadata/

1.4 About Partitioned Models

Introduces partitioned models to organize and represent multiple models.

When you build a model on your data set and apply it to new data, sometimes the prediction may be generic that performs badly when run on new and evolving data. To overcome this, the data set can be divided into different parts based on some characteristics. Oracle Machine Learning for SQL supports partitioned model. Partitioned models allow users to build a type of ensemble model for each data partition. The top-level model has sub models that are automatically produced. The sub models are based on the attribute options. For example, if your data set has an attribute called `REGION` with four values and you have defined it as the partitioned attribute. Then, four sub models are created for this attribute. The sub models are automatically managed and used as a single model. The partitioned model automates a typical machine learning task and can potentially achieve better accuracy through multiple targeted models.

The partitioned model and its sub models reside as first class, persistent database objects. Persistent means that the partitioned model has an on-disk representation. In a partition model, the performance of partitioned models with a large number of partitions is enhanced, and dropping a single model within a partition model is also improved.

To create a partitioned model, include the `ODMS_PARTITION_COLUMNS` setting. To define the number of partitions, include the `ODMS_MAX_PARTITIONS` setting. When you are making predictions, you must use the top-level model. The correct sub model is selected automatically based on the attribute, the attribute options, and the partition setting. You must include the partition columns as part of the `USING` clause when scoring. The `GROUPING` hint is an optional hint that applies to machine learning scoring functions when scoring partitioned models.

The partition names, key values, and the structure of the partitioned model are available in the `ALL_MINING_MODEL_PARTITIONS` view.

Related Topics

- *Oracle Database Reference*

> **See Also:**
>
> *Oracle Database SQL Language Reference* on how to use `GROUPING` hint.
> *Oracle Machine Learning for SQL User's Guide* to understand more about partitioned models.

1.5 Interfaces to Oracle Machine Learning for SQL

Introduces supported interfaces for Oracle Machine Learning for SQL.

The programmatic interfaces to Oracle Machine Learning for SQL are PL/SQL for building and maintaining models and a family of SQL functions for scoring. OML4SQL also supports a graphical user interface, which is implemented as an extension to Oracle SQL Developer.

Oracle Predictive Analytics, a set of simplified OML4SQL routines, is built on top of OML4SQL and is implemented as a PL/SQL package.

1.5.1 PL/SQL API

The OML4SQL PL/SQL API is built into the `DBMS_DATA_MINING` PL/SQL package, which has routines for building, testing, and maintaining machine learning models. This package also has a batch apply operation.

The following example shows part of a simple PL/SQL script for creating an SVM classification model called SVMC_SH_Clas_sample. The model build uses weights, specified in a weights table, and settings, specified in a settings table. The weights influence the weighting of target classes. The settings override default behavior. The model uses Automatic Data Preparation (`prep_auto_on` setting). The model is trained on the data in mining_data_build_v.

Example 1-1 Creating a Classification Model

```
-----------------------  CREATE AND POPULATE A CLASS WEIGHTS TABLE  ------------
CREATE TABLE svmc_sh_sample_class_wt (
  target_value NUMBER,
  class_weight NUMBER);
INSERT INTO svmc_sh_sample_class_wt VALUES (0,0.35);
INSERT INTO svmc_sh_sample_class_wt VALUES (1,0.65);
COMMIT;
-----------------------  CREATE AND POPULATE A SETTINGS TABLE ------------------
CREATE TABLE svmc_sh_sample_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(4000));
BEGIN
INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
  (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
  (dbms_data_mining.svms_kernel_function, dbms_data_mining.svms_linear);
INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
  (dbms_data_mining.clas_weights_table_name, 'svmc_sh_sample_class_wt');
INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
  (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_on);
END;
/
```

```
------------------------  CREATE THE MODEL -------------------------------------
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'SVMC_SH_Clas_sample',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svmc_sh_sample_settings');
END;
/
```

1.5.2 SQL Functions

Oracle Machine Learning for SQL supports SQL functions for performing prediction, clustering, and feature extraction.

The functions score data by applying an OML4SQL model object or by running an analytic clause that performs dynamic scoring.

The following example shows a query that applies the classification model `svmc_sh_clas_sample` to the data in the view `mining_data_apply_v`. The query returns the average age of customers who are likely to use an affinity card. The results are broken out by gender.

Example 1-2 The PREDICTION Function

```
SELECT cust_gender,
       COUNT(*) AS cnt,
       ROUND(AVG(age)) AS avg_age
  FROM mining_data_apply_v
 WHERE PREDICTION(svmc_sh_clas_sample USING *) = 1
GROUP BY cust_gender
ORDER BY cust_gender;


C        CNT    AVG_AGE
- ---------- ----------
F         59         41
M        409         45
```

Related Topics

- In-Database Scoring
  Scoring is the application of a machine learning algorithm to new data. In Oracle Machine Learning for SQL scoring engine and the data both reside within the database.

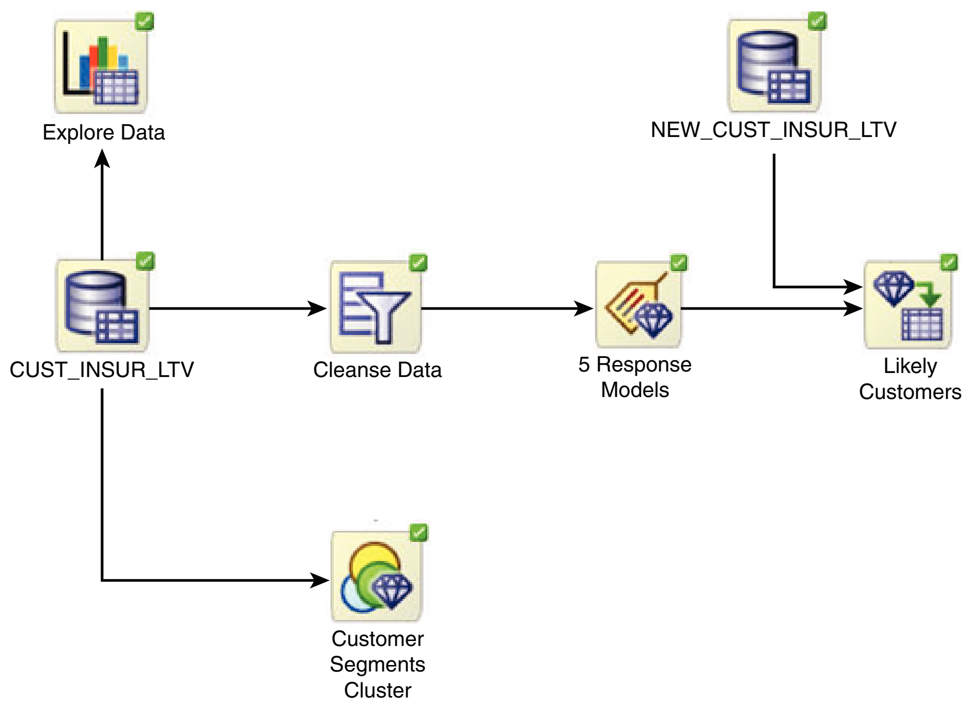
1.5.3 Oracle Data Miner

Oracle Machine Learning for SQL supports a graphical interface called Oracle Data Miner.

Oracle Data Miner is a graphical interface to OML4SQL. Oracle Data Miner is an extension to Oracle SQL Developer, which is available for download free of charge on the Oracle Technology Network.

Oracle Data Miner uses a work flow paradigm to capture, document, and automate the process of building, evaluating, and applying OML4SQL models. Within a work flow, you can specify data transformations, build and evaluate multiple models, and score multiple data sets. You can then save work flows and share them with other users.

Figure 1-1 An Oracle Data Miner Workflow



For information about Oracle Data Miner, including installation instructions, visit Oracle Technology Network.

Related Topics

- Oracle Data Miner

1.5.4 Predictive Analytics

Predictive analytics is a technology that captures Oracle Machine Learning for SQL processes in simple routines.

Sometimes called "one-click machine learning," predictive analytics simplifies and automates the machine learning process.

Predictive analytics uses OML4SQL technology, but knowledge of OML4SQL is not needed to use predictive analytics. You can use predictive analytics by specifying an operation to perform on your data. You do not need to create or use OML4SQL models or understand the OML4SQL functions and algorithms summarized in "Oracle Machine Learning for SQL Basics ".

Oracle Machine Learning for SQL predictive analytics operations are described in the following table:

Table 1-1 Oracle Predictive Analytics Operations

Operation	Description
`EXPLAIN`	Explains how individual predictors (columns) affect the variation of values in a target column
`PREDICT`	For each case (row), predicts the values in a target column
`PROFILE`	Creates a set of rules for cases (rows) that imply the same target value

The Oracle predictive analytics operations are implemented in the `DBMS_PREDICTIVE_ANALYTICS` PL/SQL package. They are also available in Oracle Data Miner.

1.6 Overview of Database Analytics

Oracle Database supports native analytical features. Since all these features are on a common server, they can be combined efficiently. Analytical results can be integrated with Oracle Business Intelligence Suite Enterprise Edition and other BI tools.

The possibilities for combining different analytics are virtually limitless. Example 1-3 shows Oracle Machine Learning for SQL and text processing within a single SQL query. The query selects all customers who have a high propensity to attrite (> 80% chance), are valuable customers (customer value rating > 90), and have had a recent conversation with customer services regarding a Checking Plus account. The propensity to attrite information is computed using a OML4SQL model called `tree_model`. The query uses the Oracle Text `CONTAINS` operator to search call center notes for references to Checking Plus accounts.

The following table shows some of the built-in analytics that Oracle Database can do:

Table 1-2 Oracle Database Native Analytics

Analytical Feature	Description	Documented In...
Complex data transformations	Data transformation is a key aspect of analytical applications and ETL (extract, transform, and load). You can use SQL expressions to implement data transformations, or you can use the `DBMS_DATA_MINING_TRANSFORM` package. `DBMS_DATA_MINING_TRANSFORM` is a flexible data transformation package that includes a variety of missing value and outlier treatments, as well as binning and normalization capabilities.	*Oracle Database PL/SQL Packages and Types Reference*
Statistical functions	Oracle Database provides a long list of SQL statistical functions with support for: hypothesis testing (such as t-test, F-test), correlation computation (such as pearson correlation), cross-tab statistics, and descriptive statistics (such as median and mode). The `DBMS_STAT_FUNCS` package adds distribution fitting procedures and a summary procedure that returns descriptive statistics for a column.	*Oracle Database SQL Language Reference* and *Oracle Database PL/SQL Packages and Types Reference*
Window and analytic SQL functions	Oracle Database supports analytic and windowing functions for computing cumulative, moving, and centered aggregates. With windowing aggregate functions, you can calculate moving and cumulative versions of `SUM`, `AVERAGE`, `COUNT`, `MAX`, `MIN`, and many more functions.	*Oracle Database Data Warehousing Guide*

Table 1-2 (Cont.) Oracle Database Native Analytics

Analytical Feature	Description	Documented In...
Linear algebra	The `UTL_NLA` package exposes a subset of the popular `BLAS` and `LAPACK` (Version 3.0) libraries for operations on vectors and matrices represented as VARRAYs. This package includes procedures to solve systems of linear equations, invert matrices, and compute eigenvalues and eigenvectors.	*Oracle Database PL/SQL Packages and Types Reference*
Analytic views	Analytic views organize data using a dimensional model. They enable you to easily add aggregations and calculations to data sets and to present data in views that can be queried with relatively simple SQL.	*Oracle Database Data Warehousing Guide*
Spatial analytics	Oracle Spatial provides advanced spatial features to support high-end GIS and LBS solutions. Oracle Spatial's analysis and machine learning capabilities include functions for binning, detection of regional patterns, spatial correlation, colocation machine learning, and spatial clustering. Oracle Spatial also includes support for topology and network data models and analytics. The topology data model of Oracle Spatial allows one to work with data about nodes, edges, and faces in a topology. It includes network analysis functions for computing shortest path, minimum cost spanning tree, nearest-neighbors analysis, traveling salesman problem, among others.	*Oracle Spatial Developer's Guide*
Graph	The Property Graph delivers advanced graph query and analytics capabilities in Oracle Database. The in-memory graph server (PGX) provides a machine learning library, which supports graph-empowered machine learning algorithms. The machine learning library supports DeepWalk, supervised GraphWise, and Pg2vec algorithms.	*Oracle Database Graph Developer's Guide for Property Graph*
Text Analysis	Oracle Text uses standard SQL to index, search, and analyze text and documents stored in the Oracle database, in files, and on the web. Oracle Text also supports automatic classification and clustering of document collections. Many of the analytical features of Oracle Text are layered on top of Oracle Machine Learning functionality.	*Oracle Text Application Developer's Guide*

Example 1-3 SQL Query Combining Oracle Machine Learning for SQL and Oracle Text

```
SELECT A.cust_name, A.contact_info
  FROM customers A
 WHERE PREDICTION_PROBABILITY(tree_model,
            'attrite' USING A.*) > 0.8
   AND A.cust_value > 90
   AND A.cust_id IN
       (SELECT B.cust_id
          FROM call_center B
         WHERE B.call_date BETWEEN '01-Jan-2005'
                               AND '30-Jun-2005'
         AND CONTAINS(B.notes, 'Checking Plus', 1) > 0);
```

2

Oracle Machine Learning Basics

Understand the basic concepts of Oracle Machine Learning.

2.1 Machine Learning Techniques

Each machine learning **technique** specifies a class of problems that can be modeled and solved.

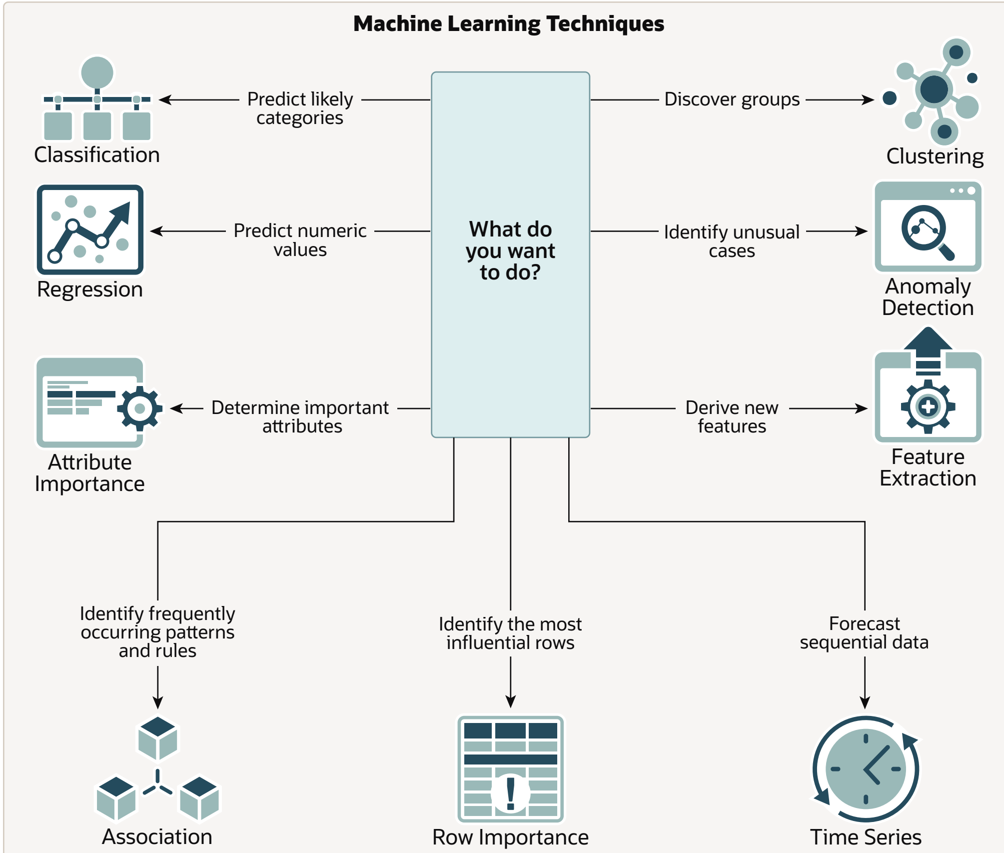
A basic understanding of machine learning techniques and algorithms is required for using Oracle Machine Learning.

Machine learning techniques fall generally into two categories: **supervised** and **unsupervised**. Notions of supervised and unsupervised learning are derived from the science of machine learning, which has been called a sub-area of artificial intelligence.

Artificial intelligence refers to the implementation and study of systems that exhibit autonomous intelligence or behavior of their own. Machine learning deals with techniques that enable devices to learn from their own performance and modify their own functioning.

The following illustration provides an idea of how to use Oracle machine learning techniques.

Figure 2-1   How to Use Machine Learning techniques



Related Topics

- What is a Machine Learning Algorithm
  An algorithm is a mathematical procedure for solving a specific kind of problem. For some machine learning techniques, you can choose among several algorithms.

2.1.1 Supervised Machine Learning

Overview of supervised machine learning.

Supervised learning is also known as directed learning. The learning process is directed by a previously known dependent attribute or target. Directed Oracle Machine Learning attempts to explain the behavior of the target as a function of a set of independent attributes or predictors.

Supervised learning generally results in predictive models. This is in contrast to unsupervised learning where the goal is pattern detection.

2.1.1.1 Supervised Learning: Testing

The process of applying the model to test data helps to determine whether the model, built on one chosen sample, is generalizable to other data. In other words, test data is used for scoring.

In particular, it helps to avoid the phenomenon of overfitting, which can occur when the logic of the model fits the build data too well and therefore has little predictive power.

2.1.1.2 Supervised Learning: Scoring

Learn about scoring in supervised learning.

Apply data, also called scoring data, is the actual population to which a model is applied. For example, you might build a model that identifies the characteristics of customers who frequently buy a certain product. To obtain a list of customers who shop at a certain store and are likely to buy a related product, you might apply the model to the customer data for that store. In this case, the store customer data is the scoring data.

Most supervised learning can be applied to a population of interest. The principal supervised machine learning techniques, **classification** and **regression**, can both be used for scoring.

Oracle Machine Learning does not support the scoring operation for **attribute importance**, another supervised technique. Models of this type are built on a population of interest to obtain information about that population; they cannot be applied to separate data. An attribute importance model returns and ranks the attributes that are most important in predicting a target value.

Oracle Machine Learning supports the supervised machine learning techniques described in the following table:

Table 2-1 Oracle Machine Learning Supervised Techniques

Technique	Description	Sample Problem
Attribute Importance	Identifies the attributes that are most important in predicting a target attribute	Given customer response to an affinity card program, find the most significant predictors
Classification	Assigns items to discrete classes and predicts the class to which an item belongs	Given demographic data about a set of customers, predict customer response to an affinity card program
Regression	Approximates and forecasts continuous values	Given demographic and purchasing data about a set of customers, predict customers' age

2.1.2 Unsupervised Machine Learning

Overview of unsupervised machine learning.

Unsupervised learning is non-directed. There is no distinction between dependent and independent attributes. There is no previously-known result to guide the algorithm in building the model.

Unsupervised learning can be used for **descriptive** purposes. It can also be used to make predictions.

2.1.2.1 Unsupervised Learning: Scoring

Introduces unsupervised learning, supported scoring operations, and unsupervised machine learning techniques.

Although unsupervised machine learning does not specify a target, most unsupervised learning can be applied to a population of interest. For example, clustering models use descriptive machine learning techniques, but they can be applied to classify cases according to their cluster assignments. **Anomaly Detection**, although unsupervised, is typically used to predict whether a data point is typical among a set of cases.

Oracle Machine Learning supports the scoring operation for **Clustering** and **Feature Extraction**, both unsupervised machine learning techniques. Oracle Machine Learning does not support the scoring operation for **Association Rules**, another unsupervised function. Association models are built on a population of interest to obtain information about that population; they cannot be applied to separate data. An association model returns rules that explain how items or events are associated with each other. The association rules are returned with statistics that can be used to rank them according to their probability.

OML supports the unsupervised techniques described in the following table:

Table 2-2 Oracle Machine Learning Unsupervised Techniques

Function	Description	Sample Problem
Anomaly Detection	Identifies items (outliers) that do not satisfy the characteristics of "normal" data	Given demographic data about a set of customers, identify customer purchasing behavior that is significantly different from the norm
Association Rules	Finds items that tend to co-occur in the data and specifies the rules that govern their co-occurrence	Find the items that tend to be purchased together and specify their relationship
Clustering	Finds natural groupings in the data	Segment demographic data into clusters and rank the probability that an individual belongs to a given cluster
Feature Extraction	Creates new attributes (features) using linear combinations of the original attributes	Given demographic data about a set of customers, group the attributes into general characteristics of the customers

Related Topics

- Machine Learning Techniques
  Part II provides basic conceptual information about machine learning techniques that the Oracle Machine Learning for SQL supports.
- In-Database Scoring
  Scoring is the application of a machine learning algorithm to new data. In Oracle Machine Learning for SQL scoring engine and the data both reside within the database.

2.2 What is a Machine Learning Algorithm

An algorithm is a mathematical procedure for solving a specific kind of problem. For some machine learning techniques, you can choose among several algorithms.

Each algorithm produces a specific type of model, with different characteristics. Some machine learning problems can best be solved by using more than one algorithm in combination. For example, you might first use a feature extraction model to create an optimized set of predictors, then a classification model to make a prediction on the results.

2.2.1 Oracle Machine Learning Supervised Algorithms

Oracle Machine Learning for SQL (OML4SQL) supports the supervised machine learning algorithms described in the following table.

Table 2-3 Oracle Machine Learning Algorithms for Supervised techniques

Algorithm	Technique	Description
Decision Tree	Classification	Decision trees extract predictive information in the form of human-understandable rules. The rules are if-then-else expressions; they explain the decisions that lead to the prediction.
Explicit Semantic Analysis	Classification	Explicit Semantic Analysis (ESA) is designed to make predictions for text data. This algorithm can address use cases with hundreds of thousands of classes. In Oracle Database 12*c* Release 2, ESA was introduced as Feature Extraction algorithm.
Exponential Smoothing	Time Series and Time Series Regression	Exponential Smoothing (ESM) provides forecasts for time series data. Forecasts are made for each time period within a user-specified forecast window. ESM provides a total of 14 different time series models, including all the most popular estimates of trend and seasonal effects. Choice of model is controlled by user settings. ESM provides confidence bounds on its forecasts.
Generalized Linear Model	Classification and Regression	Generalized Linear Model (GLM) implements logistic regression for classification of binary targets and linear regression for continuous targets. GLM classification supports confidence bounds for prediction probabilities. GLM regression supports confidence bounds for predictions.
Minimum Description Length	Attribute Importance	Minimum Description Length (MDL) is an information theoretic model selection principle. MDL assumes that the simplest, most compact representation of data is the best and most probable explanation of the data.
Naive Bayes	Classification	Naive Bayes makes predictions using Bayes' Theorem, which derives the probability of a prediction from the underlying evidence, as observed in the data.
Neural Network	Classification and Regression	Neural Network in machine learning is an artificial algorithm inspired from biological neural network and is used to estimate or approximate Techniques that depend on a large number of generally unknown inputs. Neural Network is designed for classification and regression.
Random Forest	Classification	Random Forest is a powerful machine learning algorithm. The Random Forest algorithm builds a number of Decision Tree models and predicts using the ensemble of trees.
Support Vector Machine	Classification and Regression	Distinct versions of the Support Vector Machine (SVM) algorithm use different kernel Techniques to handle different types of data sets. Linear and Gaussian (nonlinear) kernels are supported. SVM classification attempts to separate the target classes with the widest possible margin. SVM regression tries to find a continuous function such that the maximum number of data points lie within an epsilon-wide tube around it.
XGBoost	Classification and Regression	XGBoost is machine learning algorithm for regression and classification that makes available the XGBoost open source package. Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

2.2.2 Oracle Machine Learning Unsupervised Algorithms

Oracle Machine Learning for SQL (OML4SQL) supports the unsupervised machine learning algorithms described in the following table.

Table 2-4 Oracle Machine Learning Algorithms for Unsupervised Techniques

Algorithm	Technique	Description
Apriori	Association	Apriori performs market basket analysis by identifying co-occurring items (frequent itemsets) within a set. Apriori finds rules with support greater than a specified minimum support and confidence greater than a specified minimum confidence.
CUR Matrix Decomposition	Attribute Importance	CUR Matrix Decomposition is an alternative to Support Vector Machine (SVM) and Principal Component Analysis (PCA) and an important tool for exploratory data analysis. This algorithm performs analytical processing and singles out important columns and rows.
Expectation Maximization	Clustering and Anomaly Detection	Expectation Maximization (EM) is a density estimation algorithm that performs probabilistic clustering. In density estimation, the goal is to construct a density function that captures how a given population is distributed. The density estimate is based on observed data that represents a sample of the population. Oracle Machine Learning supports probabilistic clustering and data frequency estimates and other applications of Expectation Maximization. The EM Anomaly algorithm can detect the underlying data distribution and thereby identify records that do not fit the learned data distribution well.
Explicit Semantic Analysis	Feature Extraction	Explicit Semantic Analysis (ESA) uses existing knowledge base as features. An attribute vector represents each feature or a concept. ESA creates a reverse index that maps every attribute to the knowledge base concepts or the concept-attribute association vector value.
k-Means	Clustering	*k*-Means is a distance-based clustering algorithm that partitions the data into a predetermined number of clusters. Each cluster has a centroid (center of gravity). Cases (individuals within the population) that are in a cluster are close to the centroid. OML4SQL supports an enhanced version of *k*-Means. It goes beyond the classical implementation by defining a hierarchical parent-child relationship of clusters.
Multivariate State Estimation Technique - Sequential Probability Ratio Test	Anomaly Detection	The Multivariate State Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT) algorithm is a nonlinear, nonparametric anomaly detection machine learning technique designed for monitoring critical processes. It detects subtle anomalies while also producing minimal false alarms.
Non-Negative Matrix Factorization	Feature Extraction	Non-Negative Matrix Factorization (NMF) generates new attributes using linear combinations of the original attributes. The coefficients of the linear combinations are non-negative. During model apply, an NMF model maps the original data into the new set of attributes (features) discovered by the model.
One Class Support Vector Machine	Anomaly Detection	One-class SVM builds a profile of one class. When the model is applied, it identifies cases that are somehow different from that profile. This allows for the detection of rare cases that are not necessarily related to each other.

Table 2-4 (Cont.) Oracle Machine Learning Algorithms for Unsupervised Techniques

Algorithm	Technique	Description
Orthogonal Partitioning Clustering	Clustering	Orthogonal Partitioning Clustering (O-Cluster) creates a hierarchical, grid-based clustering model. The algorithm creates clusters that define dense areas in the attribute space. A sensitivity parameter defines the baseline density level.
Singular Value Decomposition and Principal Component Analysis	Feature Extraction	Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) are orthogonal linear transformations that are optimal at capturing the underlying variance of the data. This property is extremely useful for reducing the dimensionality of high-dimensional data and for supporting meaningful data visualization. In addition to dimensionality reduction, SVD and PCA have a number of other important applications, such as data de-noising (smoothing), data compression, matrix inversion, and solving a system of linear equations.

Related Topics

- Algorithms
  Oracle Machine Learning for SQL supports the algorithms listed in Part III. Part III provides basic conceptual information about the algorithms. There is at least one algorithm for each of the machine learning techniques.

2.3 Data Preparation

Preparing the data is a valuable step in solving machine learning problems.

The quality of a model depends to a large extent on the quality of the data used to build (train) it. Much of the time spent in any given machine learning project is devoted to data preparation. The data must be carefully inspected, cleansed, and transformed, and algorithm-appropriate data preparation methods must be applied.

The process of data preparation is further complicated by the fact that any data to which a model is applied, whether for testing or for scoring, must undergo the same transformations as the data used to train the model.

2.3.1 Simplify Data Preparation with Oracle Machine Learning for SQL

Oracle Machine Learning for SQL (OML4SQL) provides inbuilt data preparation, automatic data preparation, custom data preparation through the `DBMS_DATA_MINING_TRANSFORM` PL/SQL package, model details, and employs consistent approach across machine learning algorithms to manage missing and sparse data.

OML4SQL offers several features that significantly simplify the process of data preparation:

- Embedded data preparation: The transformations used in training the model are embedded in the model and automatically run whenever the model is applied to new data. If you specify transformations for the model, you only have to specify them once.
- Automatic Data Preparation (ADP): Oracle Machine Learning for SQL supports an automated data preparation mode. When ADP is active, Oracle Machine Learning for SQL automatically performs the data transformations required by the algorithm. The transformation instructions are embedded in the model along with any user-specified transformation instructions.

- Automatic management of missing values and sparse data: Oracle Machine Learning for SQL uses consistent methodology across machine learning algorithms to handle sparsity and missing values.
- Transparency: Oracle Machine Learning for SQL provides model details, which are a view of the attributes that are internal to the model. This insight into the inner details of the model is possible because of reverse transformations, which map the transformed attribute values to a form that can be interpreted by a user. Where possible, attribute values are reversed to the original column values. Reverse transformations are also applied to the target of a supervised model, thus the results of scoring are in the same units as the units of the original target.
- Tools for custom data preparation: Oracle Machine Learning for SQL provides many common transformation routines in the `DBMS_DATA_MINING_TRANSFORM` PL/SQL package. You can use these routines, or develop your own routines in SQL, or both. The SQL language is well suited for implementing transformations in the database. You can use custom transformation instructions along with ADP or instead of ADP.

2.3.2 Case Data

Learn the importance of case data in machine learning.

Most machine learning algorithms act on single-record case data, where the information for each case is stored in a separate row. The data attributes for the cases are stored in the columns.

When the data is organized in transactions, the data for one case (one transaction) is stored in many rows. An example of transactional data is market basket data. With the single exception of Association Rules, which can operate on native transactional data, Oracle Machine Learning for SQL algorithms require single-record case organization.

2.3.2.1 Nested Data

Learn how nested columns are treated in Oracle Machine Learning for SQL.

OML4SQL supports attributes in nested columns. A transactional table can be cast as a nested column and included in a table of single-record case data. Similarly, star schemas can be cast as nested columns. With nested data transformations, Oracle Machine Learning for SQL can effectively mine data originating from multiple sources and configurations.

2.3.3 Text Data

Prepare and transform unstructured text data for machine learning.

Oracle Machine Learning for SQL interprets `CLOB` columns and long `VARCHAR2` columns automatically as unstructured text. Additionally, you can specify columns of short `VARCHAR2`, `CHAR`, `BLOB`, and `BFILE` as unstructured text. Unstructured text includes data items such as web pages, document libraries, Power Point presentations, product specifications, emails, comment fields in reports, and call center notes.

OML4SQL uses Oracle Text utilities and term weighting strategies to transform unstructured text for analysis. In text transformation, text terms are extracted and given numeric values in a text index. The text transformation process is configurable for the model and for individual attributes. Once transformed, the text can by mined with a OML4SQL algorithm.

Related Topics

- Prepare the Data
- Machine Learning Operations on Unstructured Text

2.4 In-Database Scoring

Scoring is the application of a machine learning algorithm to new data. In Oracle Machine Learning for SQL scoring engine and the data both reside within the database.

In traditional machine learning, models are built using specialized software on a remote system and deployed to another system for scoring. This is a cumbersome, error-prone process open to security violations and difficulties in data synchronization.

With OML4SQL, scoring is simple and secure. The scoring engine and the data both reside within the database. Scoring is an extension to the SQL language, so the results of machine learning can easily be incorporated into applications and reporting systems.

2.4.1 Parallel Execution and Ease of Administration

All Oracle Machine Learning for SQL scoring routines support parallel execution for scoring large data sets.

In-database scoring provides performance advantages. All Oracle Machine Learning for SQL scoring routines support parallel execution, which significantly reduces the time required for executing complex queries and scoring large data sets.

In-database machine learning minimizes the IT effort needed to support OML4SQL initiatives. Using standard database techniques, models can easily be refreshed (re-created) on more recent data and redeployed. The deployment is immediate since the scoring query remains the same; only the underlying model is replaced in the database.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*

2.4.2 SQL Functions for Model Apply and Dynamic Scoring

In Oracle Machine Learning for SQL, scoring is performed by SQL language functions. Understand the different ways of scoring using SQL functions.

The functions perform prediction, clustering, and feature extraction. The functions can be invoked in two different ways: By applying a machine learning model object (Example 2-1), or by executing an analytic clause that computes the machine learning analysis dynamically and applies it to the data (Example 2-2). Dynamic scoring, which eliminates the need for a model, can supplement, or even replace, the more traditional methodology described in "The Machine Learning Process".

In Example 2-1, the `PREDICTION_PROBABILITY` function applies the model svmc_sh_clas_sample, created in Example 1-1, to score the data in `mining_data_apply_v`. The function returns the ten customers in Italy who are most likely to use an affinity card.

In Example 2-2, the functions `PREDICTION` and `PREDICTION_PROBABILITY` use the analytic syntax (the `OVER` () clause) to dynamically score the data in `mining_data_apply_v`. The query returns the customers who currently do not have an affinity card with the probability that they are likely to use.

Example 2-1 Applying a Oracle Machine Learning for SQL Model to Score Data

```
SELECT cust_id FROM
  (SELECT cust_id,
        rank() over (order by PREDICTION_PROBABILITY(svmc_sh_clas_sample, 1
                     USING *) DESC, cust_id) rnk
   FROM mining_data_apply_v
   WHERE country_name = 'Italy')
WHERE rnk <= 10
ORDER BY rnk;



 CUST_ID
----------
    101445
    100179
    100662
    100733
    100554
    100081
    100344
    100324
    100185
    101345
```

Example 2-2 Executing an Analytic Function to Score Data

```
SELECT cust_id, pred_prob FROM
  (SELECT cust_id, affinity_card,
    PREDICTION(FOR TO_CHAR(affinity_card) USING *) OVER () pred_card,
    PREDICTION_PROBABILITY(FOR TO_CHAR(affinity_card),1 USING *) OVER ()
pred_prob
   FROM mining_data_build_v)
WHERE affinity_card = 0
AND pred_card = 1
ORDER BY pred_prob DESC;



 CUST_ID PRED_PROB
---------- ---------
    102434       .96
    102365       .96
    102330       .96
    101733       .95
    102615       .94
    102686       .94
    102749       .93
    .
    .
```

```
.
101656       .51
```

Related Topics

- *Oracle Database SQL Language Reference*
- *Oracle Machine Learning for SQL User’s Guide*

Part II

Machine Learning Techniques

Part II provides basic conceptual information about machine learning techniques that the Oracle Machine Learning for SQL supports.

Machine learning techniques represent a class of problems that can be solved using OML4SQL algorithms.

> **Note:**
>
> The term machine learning technique has no relationship to a SQL language function.

Part II contains these chapters:

- Anomaly Detection
- Association
- Classification
- Clustering
- Embedding
- Feature Extraction
- Feature Selection
- Ranking
- Regression
- Row Importance
- Time Series

Related Topics

- Algorithms
  Oracle Machine Learning for SQL supports the algorithms listed in Part III. Part III provides basic conceptual information about the algorithms. There is at least one algorithm for each of the machine learning techniques.
- *Oracle Database SQL Language Reference*

3 Regression

Learn how to predict a continuous numerical target through regression - the supervised machine learning technique.

3.1 About Regression

Regression is an Oracle Machine Learning for SQL function that predicts numeric values along a continuum.

Profit, sales, mortgage rates, house values, square footage, temperature, or distance can be predicted using Regression techniques. For example, a regression model can be used to predict the value of a house based on location, number of rooms, lot size, and other factors.

A regression task begins with a data set in which the target values are known. For example, a regression model that predicts house values can be developed based on observed data for many houses over a period of time. In addition to the value, the data can track the age of the house, square footage, number of rooms, taxes, school district, proximity to shopping centers, and so on. House value can be the target, the other attributes are the predictors, and the data for each house constitutes a case.

In the model build (training) process, a regression algorithm estimates the value of the target as a function of the predictors for each case in the build data. These relationships between predictors and target are summarized in a model, which can then be applied to a different data set in which the target values are unknown.

Regression models are tested by computing various statistics that measure the difference between the predicted values and the expected values. The historical data for a regression project is typically divided into two data sets: one for building the model, the other for testing the model.

Regression modeling has many applications in trend analysis, business planning, marketing, financial forecasting, time series prediction, biomedical and drug response modeling, and environmental modeling.

3.1.1 How Does Regression Work?

Understand regression as a mathematical expression.

You do not need to understand the mathematics used in regression analysis to develop and use quality regression models for Oracle Machine Learning for SQL. However, it is helpful to understand a few basic concepts.

Regression analysis seeks to determine the values of parameters for a function that cause the function to best fit a set of data observations that you provide. The following equation expresses these relationships in symbols. It shows that regression is the process of estimating the value of a continuous target (*y*) as a function (*F*) of one or more predictors ($\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$), a set of parameters (θ_1 , θ_2 , ..., θ_n), and a measure of error (*e*).

$$y = F(\mathbf{x},\theta) \quad + e$$

The predictors can be understood as independent variables and the target as a dependent variable. The error, also called the **residual**, is the difference between the expected and predicted value of the dependent variable. The regression parameters are also known as **regression coefficients**.

The process of training a regression model involves finding the parameter values that minimize a measure of the error, for example, the sum of squared errors.

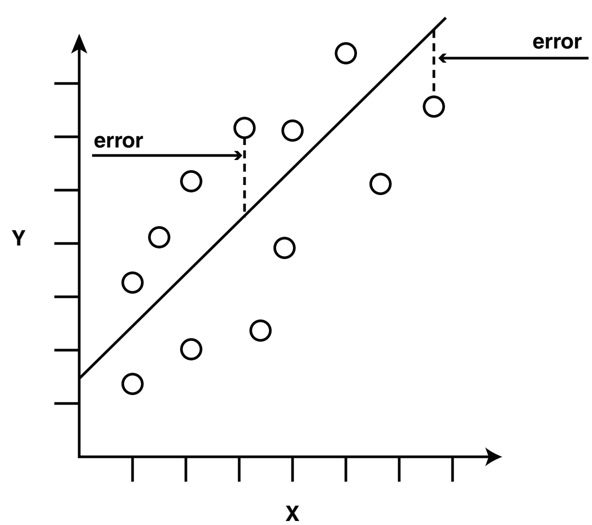
There are different families of regression functions and different ways of measuring the error.

3.1.1.1 Linear Regression

A linear regression technique can be used if the relationship between the predictors and the target can be approximated with a straight line.

Regression with a single predictor is the easiest to visualize. Simple linear regression with a single predictor is shown in the following figure:

Figure 3-1   Linear Regression With a Single Predictor



Linear regression with a single predictor can be expressed with the following equation.

$$y = \theta_2\mathbf{x} \quad + \theta_1 \quad + e$$

The regression parameters in simple linear regression are:

- The **slope** of the line ($_2$) — the angle between a data point and the regression line
- The ***y* intercept** ($_1$) — the point where **x** crosses the *y* axis (**x** = 0)

3.1.1.2 Multivariate Linear Regression

The term **multivariate linear regression** refers to linear regression with two or more predictors ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$). When multiple predictors are used, the regression line cannot be visualized in two-dimensional space. However, the line can be computed by expanding the equation for single-predictor linear regression to include the parameters for each of the predictors.

$$y = \theta_1 + \theta_2\mathbf{x}_1 + \theta_3\mathbf{x}_2 + \ldots\ldots \theta_n \mathbf{x}_{n-1} + e$$

3.1.1.3 Regression Coefficients

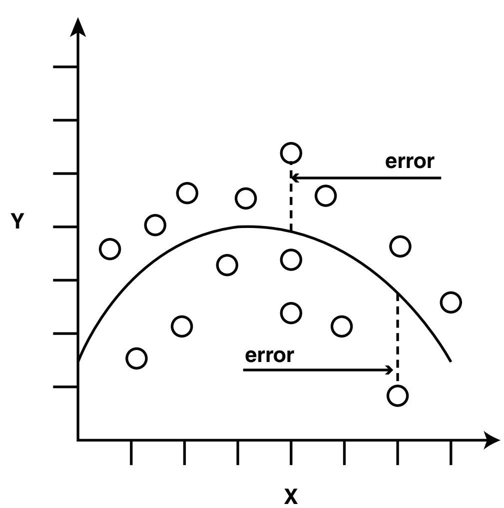
In multivariate linear regression, the regression parameters are often referred to as coefficients. When you build a multivariate linear regression model, the algorithm computes a coefficient for each of the predictors used by the model. The coefficient is a measure of the impact of the predictor **x** on the target *y*. Numerous statistics are available for analyzing the regression coefficients to evaluate how well the regression line fits the data.

3.1.1.4 Nonlinear Regression

Often the relationship between **x** and ***y*** cannot be approximated with a straight line. In this case, a nonlinear regression technique can be used. Alternatively, the data can be preprocessed to make the relationship linear.

Nonlinear regression models define ***y*** as a function of **x** using an equation that is more complicated than the linear regression equation. In the following figure, **x** and ***y*** have a nonlinear relationship.

Figure 3-2   Nonlinear Regression With a Single Predictor



3.1.1.5 Multivariate Nonlinear Regression

The term **multivariate nonlinear regression** refers to nonlinear regression with two or more predictors ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$). When multiple predictors are used, the nonlinear relationship cannot be visualized in two-dimensional space.

3.1.1.6 Confidence Bounds

A regression model predicts a numeric target value for each case in the scoring data. In addition to the predictions, some regression algorithms can identify confidence bounds, which are the upper and lower boundaries of an interval in which the predicted value is likely to lie.

When a model is built to make predictions with a given confidence, the confidence interval is produced along with the predictions. For example, a model predicts the value of a house to be $500,000 with a 95% confidence that the value is between $475,000 and $525,000.

3.2 Testing a Regression Model

Apply a regression model to test data, compare predicted values with actual ones, and use metrics to evaluate accuracy.

A regression model is tested by applying it to test data with known target values and comparing the predicted values with the known values.

The test data must be compatible with the data used to build the model and must be prepared in the same way that the build data was prepared. Typically the build data and test data come from the same historical data set. A percentage of the records is used to build the model; the remaining records are used to test the model.

Test metrics are used to assess how accurately the model predicts these known values. If the model performs well and meets the business requirements, it can then be applied to new data to predict the future.

3.2.1 Regression Statistics

The Root Mean Squared Error and the Mean Absolute Error are commonly used statistics for evaluating the overall quality of a regression model. Different statistics may also be available depending on the regression methods used by the algorithm.

3.2.1.1 Root Mean Squared Error

The Root Mean Squared Error (RMSE) is the square root of the average squared distance of a data point from the fitted line.

This SQL expression calculates the RMSE.

```
SQRT(AVG((predicted_value - actual_value) * (predicted_value -
actual_value)))
```

This formula shows the RMSE in mathematical symbols. The large sigma character represents summation; *j* represents the current predictor, and *n* represents the number of predictors.

Figure 3-3 Room Mean Squared Error

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

3.2.1.2 Mean Absolute Error

The Mean Absolute Error (MAE) is the average of the absolute value of the residuals (error). The MAE is very similar to the RMSE but is less sensitive to large errors.

This SQL expression calculates the MAE.

```
AVG(ABS(predicted_value - actual_value))
```

This formula shows the MAE in mathematical symbols. The large sigma character represents summation; *j* represents the current predictor, and *n* represents the number of predictors.

Figure 3-4 Mean Absolute Error

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

3.3 Regression Algorithms

Oracle Machine Learning for SQL supports these algorithms for regression: Generalized Linear Model (GLM), Neural Network (NN), Support Vector Machine (SVM), and XGBoost.

GLM and SVM algorithms are particularly suited for analysing data sets that have very high dimensionality (many attributes), including transactional and unstructured data.

- **Generalized Linear Model**

  GLM is a popular statistical technique for linear modeling. Oracle Machine Learning for SQL implements GLM for regression and for binary classification. GLM provides extensive coefficient statistics and model statistics, as well as row diagnostics. GLM also supports confidence bounds.

- **Neural Network**

  Neural Network is a powerful algorithm that can learn arbitrary nonlinear regression functions.

- **Support Vector Machine**

  SVM is a powerful, state-of-the-art algorithm for linear and nonlinear regression. OML4SQL implements SVM for regression, classification, and anomaly detection. SVM regression supports two kernels: the Gaussian kernel for nonlinear regression and the linear kernel for linear regression.

> **Note:**
>
> OML4SQL uses the linear kernel SVM as the default regression algorithm.

- **XGBoost**

  XGBoost is machine learning algorithm for regression and classification that makes available the XGBoost open source package. Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

4

Classification

Learn how to predict a categorical target through classification - the supervised machine learning technique.

4.1 About Classification

Classification is a machine learning technique that assigns items in a collection to target categories or classes.

The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model can be used to identify loan applicants as low, medium, or high credit risks.

A classification task begins with a data set in which the class assignments are known. For example, a classification model that predicts credit risk can be developed based on observed data for many loan applicants over a period of time. In addition to the historical credit rating, the data might track employment history, home ownership or rental, years of residence, number and type of investments, and so on. Credit rating is the target, the other attributes are the predictors, and the data for each customer constitutes a case.

Classification are discrete and do not imply order. Continuous, floating-point values indicate a numerical, rather than a categorical, target. A predictive model with a numerical target uses a regression algorithm, not a classification algorithm.

The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values: for example, high credit rating or low credit rating. Multiclass targets have more than two values: for example, low, medium, high, or unknown credit rating.

In the model build (training) process, a classification algorithm finds relationships between the values of the predictors and the values of the target. Different classification algorithms use different techniques for finding relationships. These relationships are summarized in a model, which can then be applied to a different data set in which the class assignments are unknown.

Classification models are tested by comparing the predicted values to known target values in a set of test data. The historical data for a classification project is typically divided into two data sets: one for building the model; the other for testing the model.

Applying a classification model results in class assignments and probabilities for each case. For example, a model that classifies customers as low, medium, or high value also predicts the probability of each classification for each customer.

Classification has many applications in customer segmentation, business modeling, marketing, credit analysis, and biomedical and drug response modeling.

4.2 Testing a Classification Model

Test a classification model by applying it to compatible test data, comparing predictions with actuals, and assessing accuracy with test metrics.

A classification model is tested by applying it to test data with known target values and comparing the predicted values with the known values.

The test data must be compatible with the data used to build the model and must be prepared in the same way that the build data was prepared. Typically the build data and test data come from the same historical data set. A percentage of the records is used to build the model; the remaining records are used to test the model.

Test metrics are used to assess how accurately the model predicts the known values. If the model performs well and meets the business requirements, it can then be applied to new data to predict the future.

4.2.1 Confusion Matrix

A confusion matrix displays the number of correct and incorrect predictions made by the model compared with the actual classifications in the test data. The matrix is *n*-by-*n*, where *n* is the number of classes.

The following figure shows a confusion matrix for a binary classification model. The rows present the number of actual classifications in the test data. The columns present the number of predicted classifications made by the model.

Figure 4-1 Confusion Matrix for a Binary Classification Model

		PREDICTED CLASS	
		affinity_card = 1	**affinity_card = 0**
ACTUAL CLASS	**affinity_card = 1**	**516**	**25**
	affinity_card = 0	**10**	**725**

In this example, the model correctly predicted the positive class (also called true positive (TP)) for `affinity_card` 516 times and incorrectly predicted (also called false negative (FN)) it 25 times. The model correctly predicted the negative class (also called true negative (TN)) for `affinity_card` 725 times and incorrectly predicted (also called false positive (FP)) it 10 times. The following can be computed from this confusion matrix:

- The model made 1241 correct predictions, that is, TP + TN, (516 + 725).
- The model made 35 incorrect predictions, that is, FN + FP, (25 + 10).
- There are 1276 total scored cases, (516 + 25 + 10 + 725).
- The error rate is 35/1276 = 0.0274. (FN+FP/Total)
- The overall accuracy rate is 1241/1276 = 0.9725 (TP+TN)/Total).

Precision and Recall

Consider the same example, the accuracy rate shows 0.97. However, there are cases where the model has incorrectly predicted. **Precision** (positive predicted value) is the ability of a classification model to return only relevant cases. Precision can be calculated as TP/TP+FP. **Recall** (sensitivity or true positive rate) is the ability of a classification model to return relevant cases. Recall can be calculated as TP/TP+FN. The precision in this example is 516/526 = 0.98. The recall in this example is 516/541 = 0.95. Ideally, the model is good when both precision and recall are 1. This can happen when the numerator and the denominator are equal. That means, for precision, FP is zero and for recall, FN is zero.

4.2.2 Lift

Lift measures the degree to which the predictions of a classification model are better than randomly-generated predictions.

Lift applies to binary classification only, and it requires the designation of a positive class. If the model itself does not have a binary target, you can compute lift by designating one class as positive and combining all the other classes together as one negative class.

Numerous statistics can be calculated to support the notion of lift. Basically, lift can be understood as a ratio of two percentages: the percentage of correct positive classifications made by the model to the percentage of actual positive classifications in the test data. For example, if 40% of the customers in a marketing survey have responded favorably (the positive classification) to a promotional campaign in the past and the model accurately predicts 75% of them, the lift is obtained by dividing .75 by .40. The resulting lift is 1.875.

Lift is computed against quantiles that each contain the same number of cases. The data is divided into quantiles after it is scored. It is ranked by probability of the positive class from highest to lowest, so that the highest concentration of positive predictions is in the top quantiles. A typical number of quantiles is 10.

Lift is commonly used to measure the performance of response models in marketing applications. The purpose of a response model is to identify segments of the population with potentially high concentrations of positive responders to a marketing campaign. Lift reveals how much of the population must be solicited to obtain the highest percentage of potential responders.

Related Topics

- Positive and Negative Classes
  Discusses the importance of positive and negative classes in a confusion matrix.

4.2.2.1 Lift Statistics

Learn the different Lift statistics that Oracle Machine Learning for SQL can compute.

Oracle Machine Learning for SQL computes the following lift statistics:

- **Probability threshold** for a quantile *n* is the minimum probability for the positive target to be included in this quantile or any preceding quantiles (quantiles *n*-1, *n*-2,..., 1). If a cost matrix is used, a cost threshold is reported instead. The cost threshold is the maximum cost for the positive target to be included in this quantile or any of the preceding quantiles.
- **Cumulative gain** is the ratio of the cumulative number of positive targets to the total number of positive targets.
- **Target density** of a quantile is the number of true positive instances in that quantile divided by the total number of instances in the quantile.
- **Cumulative target density** for quantile *n* is the target density computed over the first *n* quantiles.
- **Quantile lift** is the ratio of the target density for the quantile to the target density over all the test data.
- **Cumulative percentage of records** for a quantile is the percentage of all cases represented by the first *n* quantiles, starting at the end that is most confidently positive, up to and including the given quantile.
- **Cumulative number of targets** for quantile *n* is the number of true positive instances in the first *n* quantiles.
- **Cumulative number of nontargets** is the number of actually negative instances in the first *n* quantiles.
- **Cumulative lift** for a quantile is the ratio of the cumulative target density to the target density over all the test data.

Related Topics

- Costs

4.2.3 Receiver Operating Characteristic (ROC)

ROC is a metric for comparing predicted and actual target values in a classification model.

ROC, like Lift, applies to binary classification and requires the designation of a positive class.

You can use ROC to gain insight into the decision-making ability of the model. How likely is the model to accurately predict the negative or the positive class?

ROC measures the impact of changes in the **probability threshold**. The probability threshold is the decision point used by the model for classification. The default probability threshold for binary classification is 0.5. When the probability of a prediction is 50% or more, the model predicts that class. When the probability is less than 50%, the other class is predicted. (In multiclass classification, the predicted class is the one predicted with the highest probability.)

Related Topics

- Positive and Negative Classes
  Discusses the importance of positive and negative classes in a confusion matrix.

4.2.3.1 The ROC Curve

ROC can be plotted as a curve on an X-Y axis. The **false positive rate** is placed on the X axis. The **true positive rate** is placed on the Y axis.

The top left corner is the optimal location on an ROC graph, indicating a high true positive rate and a low false positive rate.

4.2.3.2 Area Under the Curve

The area under the ROC curve (AUC) measures the discriminating ability of a binary classification model. The larger the AUC, the higher the likelihood that an actual positive case is assigned, and a higher probability of being positive than an actual negative case. The AUC measure is especially useful for data sets with unbalanced target distribution (one target class dominates the other).

4.2.3.3 ROC and Model Bias

The ROC curve for a model represents all the possible combinations of values in its confusion matrix.

Changes in the probability threshold affect the predictions made by the model. For instance, if the threshold for predicting the positive class is changed from 0.5 to 0.6, then fewer positive predictions are made. This affects the distribution of values in the confusion matrix: the number of true and false positives and true and false negatives differ.

You can use ROC to find the probability thresholds that yield the highest overall accuracy or the highest per-class accuracy. For example, if it is important to you to accurately predict the positive class, but you don't care about prediction errors for the negative class, then you can lower the threshold for the positive class. This can bias the model in favor of the positive class.

A cost matrix is a convenient mechanism for changing the probability thresholds for model scoring.

Related Topics

- Costs

4.2.3.4 ROC Statistics

Oracle Machine Learning for SQL computes the following ROC statistics:

- **Probability threshold:** The minimum predicted positive class probability resulting in a positive class prediction. Different threshold values result in different hit rates and different false alarm rates.
- **True negatives:** Negative cases in the test data with predicted probabilities strictly less than the probability threshold (correctly predicted).
- **True positives:** Positive cases in the test data with predicted probabilities greater than or equal to the probability threshold (correctly predicted).
- **False negatives:** Positive cases in the test data with predicted probabilities strictly less than the probability threshold (incorrectly predicted).
- **False positives:** Negative cases in the test data with predicted probabilities greater than or equal to the probability threshold (incorrectly predicted).
- **True positive fraction**: Hit rate. (true positives/(true positives + false negatives))
- **False positive fraction:** False alarm rate. (false positives**/**(false positives + true negatives))

4.3 Biasing a Classification Model

Costs, prior probabilities, and class weights bias classification models by adjusting their decision criteria.

Costs, prior probabilities, and class weights are methods for biasing classification models.

4.3.1 Costs

A cost matrix is a mechanism for influencing the decision making of a model. A cost matrix can cause the model to minimize costly misclassifications. It can also cause the model to maximize beneficial accurate classifications.

For example, if a model classifies a customer with poor credit as low risk, this error is costly. A cost matrix can bias the model to avoid this type of error. The cost matrix can also be used to bias the model in favor of the correct classification of customers who have the worst credit history.

ROC is a useful metric for evaluating how a model behaves with different probability thresholds. You can use ROC to help you find optimal costs for a given classifier given different usage scenarios. You can use this information to create cost matrices to influence the deployment of the model.

4.3.1.1 Costs Versus Accuracy

Compares Cost matrix and Confusion matrix for costs and accuracy to evaluate model quality.

Like a confusion matrix, a cost matrix is an *n*-by-*n* matrix, where *n* is the number of classes. Both confusion matrices and cost matrices include each possible combination of actual and predicted results based on a given set of test data.

A confusion matrix is used to measure accuracy, the ratio of correct predictions to the total number of predictions. A cost matrix is used to specify the relative importance of accuracy for different predictions. In most business applications, it is important to consider costs in addition to accuracy when evaluating model quality.

Related Topics

- Confusion Matrix

4.3.1.2 Positive and Negative Classes

Discusses the importance of positive and negative classes in a confusion matrix.

The positive class is the class that you care the most about. Designation of a positive class is required for computing Lift and ROC.

In the confusion matrix, in the following figure, the value `1` is designated as the positive class. This means that the creator of the model has determined that it is more important to accurately predict customers who increase spending with an affinity card (`affinity_card`=1) than to accurately predict non-responders (`affinity_card`=0). If you give affinity cards to some customers who are not likely to use them, there is little loss to the company since the cost of the cards is low. However, if you overlook the customers who are likely to respond, you miss the opportunity to increase your revenue.

Figure 4-2 Positive and Negative Predictions

		PREDICTED CLASS	
		affinity_card = 1	**affinity_card = 0**
ACTUAL CLASS	**affinity_card = 1**	**516 (true positive)**	**25 (false negative)**
	affinity_card = 0	**10 (false positive)**	**725 (true negative)**

The true and false positive rates in this confusion matrix are:

- False positive rate — 10/(10 + 725) =.01
- True positive rate — 516/(516 + 25) =.95

Related Topics

- Lift
  Lift measures the degree to which the predictions of a classification model are better than randomly-generated predictions.

- Receiver Operating Characteristic (ROC)
  ROC is a metric for comparing predicted and actual target values in a classification model.

4.3.1.3 Assigning Costs and Benefits

In a cost matrix, positive numbers (costs) can be used to influence negative outcomes. Since negative costs are interpreted as benefits, negative numbers (benefits) can be used to influence positive outcomes.

Suppose you have calculated that it costs your business $1500 when you do not give an affinity card to a customer who can increase spending. Using the model with the confusion matrix shown in Figure 4-2, each false negative (misclassification of a responder) costs $1500. Misclassifying a non-responder is less expensive to your business. You estimate that each false positive (misclassification of a non-responder) only costs $300.

You want to keep these costs in mind when you design a promotion campaign. You estimate that it costs $10 to include a customer in the promotion. For this reason, you associate a benefit of $10 with each true negative prediction, because you can eliminate those customers from your promotion. Each customer that you eliminate represents a savings of $10. In your cost matrix, you specify this benefit as -10, a negative cost.

The following figure shows how you would represent these costs and benefits in a cost matrix:

Figure 4-3 Cost Matrix Representing Costs and Benefits

		PREDICTED	
		affinity_card = 1	**affinity_card = 0**
ACTUAL	**affinity_card = 1**	**0**	**1500**
	affinity_card = 0	**300**	**-10**

With Oracle Machine Learning for SQL you can specify costs to influence the scoring of any classification model. Decision Tree models can also use a cost matrix to influence the model build.

4.3.2 Priors and Class Weights

Learn about Priors and Class Weights in a classification model to produce a useful result.

With Bayesian models, you can specify **Prior** probabilities to offset differences in distribution between the build data and the real population (scoring data). With other forms of classification, you are able to specify **Class Weights**, which have the same biasing effect as priors.

In many problems, one target value dominates in frequency. For example, the positive responses for a telephone marketing campaign is 2% or less, and the occurrence of fraud in credit card transactions is less than 1%. A classification model built on historic data of this type cannot observe enough of the rare class to be able to distinguish the characteristics of the two classes; the result can be a model that when applied to new data predicts the frequent class for every case. While such a model can be highly accurate, it is not be very useful. This illustrates that it is not a good idea to rely solely on accuracy when judging the quality of a classification model.

To correct for unrealistic distributions in the training data, you can specify priors for the model build process. Other approaches to compensating for data distribution issues include stratified sampling and anomaly detection.

Related Topics

- About Anomaly Detection
  The goal of anomaly detection is to identify items, events, or observations that are unusual within data that is seemingly 'normal'. This data may consist of traditional enterprise data or Internet of Things (IoT) sensor data.

4.4 Classification Algorithms

Learn the different classification algorithms used in Oracle Machine Learning for SQL.

Oracle Machine Learning for SQL provides the following algorithms for classification:

- **Decision Tree**

  Decision trees automatically generate rules, which are conditional statements that reveal the logic used to build the tree.

- **Explicit Semantic Analysis**

  Explicit Semantic Analysis (ESA) is designed to make predictions for text data. This algorithm can address use cases with hundreds of thousands of classes.

- **Generalized Linear Model**

  Generalized Linear Model (GLM) is a popular statistical technique for linear modeling. OML4SQL implements GLM for binary classification and for regression. GLM provides extensive coefficient statistics and model statistics, as well as row diagnostics. GLM also supports confidence bounds.

- **Naive Bayes**

  Naive Bayes uses Bayes' Theorem, a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data.

- **Random Forest**

  Random Forest is a powerful and popular machine learning algorithm that brings significant performance and scalability benefits.

- **Support Vector Machine**

Support Vector Machine (SVM) is a powerful, state-of-the-art algorithm based on linear and nonlinear regression. OML4SQL implements SVM for binary and multiclass classification.

- **XGBoost**
  XGBoost is machine learning algorithm for regression and classification that makes available the XGBoost open source package. Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

> **Note:**
>
> OML4SQL uses Naive Bayes as the default classification algorithm.

Related Topics

- About Decision Tree
  Decision tree is a supervised machine learning algorithm used for classifying data. Decision tree has a tree structure built top-down that has a root node, branches, and leaf nodes.
- About Explicit Semantic Analysis
  In Oracle Database 12*c* Release 2, Explicit Semantic Analysis (ESA) was introduced as an unsupervised algorithm for feature extraction. Starting from Oracle Database 18c, ESA is enhanced as a supervised algorithm for classification.
- About Generalized Linear Model
  The Generalized Linear Model (GLM) includes and extends the class of linear models which address and accommodate some restrictive assumptions of the linear models.
- About Multivariate State Estimation Technique - Sequential Probability Ratio Test
  Multivariate state Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT) is an algorithm for anomaly detection and statistical testing.
- About Naive Bayes
  Naive Bayes algorithm is based on conditional probabilities. It uses Bayes' theorem, a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data.
- About Random Forest
  Random Forest is a classification algorithm that builds an **ensemble** (also called **forest**) of trees.
- About Support Vector Machine
  Support Vector Machine (SVM) is a powerful, state-of-the-art algorithm with strong theoretical foundations based on the Vapnik-Chervonenkis theory.
- About XGBoost
  Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

5

Clustering

Learn how to discover natural groupings in the data through clustering - the unsupervised machine learning technique.

5.1 About Clustering

Clustering analysis finds clusters of data objects that are similar to one another.

The members of a cluster are more like each other than they are like members of other clusters. Different clusters can have members in common. The goal of clustering analysis is to find high-quality clusters such that the inter-cluster similarity is low and the intra-cluster similarity is high.

Clustering, like classification, is used to segment the data. Unlike classification, clustering models segment data into groups that were not previously defined. Classification models segment data by assigning it to previously-defined classes, which are specified in a target. Clustering models do not use a target.

Clustering is useful for exploring data. You can use clustering algorithms to find natural groupings when there are many cases and no obvious groupings.

Clustering can serve as a useful data-preprocessing step to identify homogeneous groups on which you can build supervised models.

You can also use clustering for anomaly detection. Once you segment the data into clusters, you find that some cases do not fit well into any clusters. These cases are anomalies or outliers.

5.1.1 How are Clusters Computed?

There are several different approaches to the computation of clusters. Oracle Machine Learning for SQL supports the methods listed here.

- **Density-based**: This type of clustering finds the underlying distribution of the data and estimates how areas of high density in the data correspond to peaks in the distribution. High-density areas are interpreted as clusters. Density-based cluster estimation is probabilistic.

- **Distance-based**: This type of clustering uses a distance metric to determine similarity between data objects. The distance metric measures the distance between actual cases in the cluster and the prototypical case for the cluster. The prototypical case is known as the **centroid**.
- **Grid-based**: This type of clustering divides the input space into hyper-rectangular cells and identifies adjacent high-density cells to form clusters.

5.1.2 Scoring New Data

Although clustering is an unsupervised machine learning technique, Oracle Machine Learning for SQL supports the scoring operation for clustering.

New data is scored probabilistically.

5.1.3 Hierarchical Clustering

Oracle Machine Learning for SQL supports clustering algorithms that perform hierarchical clustering.

The leaf clusters are the final clusters generated by the algorithm. Clusters higher up in the hierarchy are intermediate clusters.

5.1.3.1 Rules

Rules describe the data in each cluster.

A rule is a conditional statement that captures the logic used to split a parent cluster into child clusters. A rule describes the conditions for a case to be assigned with some probability to a cluster.

5.1.3.2 Support and Confidence

Support and **confidence** are metrics that describe the relationships between clustering rules and cases.

Support is the percentage of cases for which the rule holds. Confidence is the probability that a case described by this rule is actually assigned to the cluster.

5.1.4 Clustering Algorithms

Learn different clustering algorithms used in Oracle Machine Learning for SQL.

Oracle Machine Learning for SQL supports these clustering algorithms:

- **Expectation Maximization**

  Expectation Maximization is a probabilistic, density-estimation clustering algorithm.
- **k-Means**

  k-Means is a distance-based clustering algorithm. OML4SQL supports an enhanced version of *k*-Means.
- **Orthogonal Partitioning Clustering (O-Cluster)**

  O-Cluster is a proprietary, grid-based clustering algorithm.

See Also:

Campos, M.M., Milenova, B.L., "O-Cluster: Scalable Clustering of Large High Dimensional Data Sets", Oracle Data Mining Technologies, 10 Van De Graaff Drive, Burlington, MA 01803.

The main characteristics of the two algorithms are compared in the following table.

Table 5-1 Clustering Algorithms Compared

Feature	k-Means	O-Cluster	Expectation Maximization
Clustering methodolgy	Distance-based	Grid-based	Distribution-based
Number of cases	Handles data sets of any size	More appropriate for data sets that have more than 500 cases. Handles large tables through active sampling	Handles data sets of any size
Number of attributes	More appropriate for data sets with a low number of attributes	More appropriate for data sets with a high number of attributes	Appropriate for data sets with many or few attributes
Number of clusters	User-specified	Automatically determined	Automatically determined
Hierarchical clustering	Yes	Yes	Yes
Probabilistic cluster assignment	Yes	Yes	Yes

Note:

OML4SQL uses *k*-Means as the default clustering algorithm.

Related Topics

- Oracle Machine Learning for SQL
- About Expectation Maximization
  Expectation maximization (EM) estimation of mixture models is a popular probability density estimation technique that is used in a variety of applications.
- About *k*-Means
  The *k*-Means algorithm is a distance-based clustering algorithm that partitions the data into a specified number of clusters.
- About O-Cluster
  O-Cluster is a fast, scalable grid-based clustering algorithm well-suited for analysing large, high-dimensional data sets. The algorithm can produce high quality clusters without relying on user-defined parameters.

5.2 Evaluating a Clustering Model

Since known classes are not used in clustering, the interpretation of clusters can present difficulties. How do you know if the clusters can reliably be used for business decision making?

Oracle Machine Learning for SQL clustering models support a high degree of model transparency. You can evaluate the model by examining information generated by the clustering algorithm: for example, the centroid of a distance-based cluster. Moreover, because the clustering process is hierarchical, you can evaluate the rules and other information related to each cluster's position in the hierarchy.

6 Anomaly Detection

Learn how to detect rare cases in the data through anomaly detection - an unsupervised function.

See Also:

- Campos, M.M., Milenova, B.L., Yarmus, J.S., "Creation and Deployment of Data Mining-Based Intrusion Detection Systems in Oracle Database 10*g*"
- K. C. Gross, V. Bhardwaj and R. Bickford, "Proactive detection of software aging mechanisms in performance critical computers," 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings., Greenbelt, MD, USA, 2002, pp. 17-23, doi: 10.1109/SEW.2002.1199445.

6.1 About Anomaly Detection

The goal of anomaly detection is to identify items, events, or observations that are unusual within data that is seemingly 'normal'. This data may consist of traditional enterprise data or Internet of Things (IoT) sensor data.

Anomaly detection is an important tool for detecting, for example, fraud, network intrusions, enterprise computing service interruptions, sensor time series prognostics, and other rare events that can have great significance but are hard to find. Anomaly detection can be used to solve problems like the following:

- A law enforcement agency compiles data about unpermitted activities, but nothing about legitimate activities. How can a suspicious activity be flagged?

  The law enforcement data is all of one class. There are no counter-examples.
- An insurance agency processes millions of insurance claims, knowing that a very small number are fraudulent. How can the fraudulent claims be identified?

  The claims data contains very few counter-examples. They are outliers.
- An IT department encounters compute resource performance anomalies. How can such anomalies be detected along with their source causes, such as resource-contention issues and complex memory leaks?

The data contains sensor output from thousands of sensors.

- An oil and gas enterprise or utility company requires proactive maintenance of business-critical assets, such as oil rigs or smart meters, to reduce operations and maintenance costs, improve up-time of revenue-generating assets, and improve safety margins for life-critical systems.

6.1.1 Anomaly Detection as a form of One-Class Classification

Anomaly detection predicts whether a data point is typical for a given distribution or not. Atypical data points can be outliers or new classes. Traditional data should only have one class, hence anomaly detection is a one-class classification.

Normally, a classification model must be trained on data that includes both examples and counterexamples for each class so that the model can learn to distinguish between them. For example, a model that predicts the side effects of a medication must be trained on data that includes a wide range of responses to the medication.

A one-class classifier develops a profile that generally describes a typical case in the training data. Deviation from the profile is identified as an anomaly. One-class classifiers are sometimes referred to as positive security models, because they seek to identify "good" behaviors and assume that all other behaviors are bad.

In single-class data, all the cases have the same classification. Counterexamples, instances of another class, are hard to specify or expensive to collect. For instance, in text document classification, it is easy to classify a document under a given topic. However, the universe of documents outside of this topic can be very large and diverse. Thus, it is not feasible to specify other types of documents as counterexamples. Anomaly detection can be used to find unusual instances of a particular type of document.

> **Note:**
>
> Solving a one-class classification problem can be difficult. The accuracy of one-class classifiers cannot usually match the accuracy of standard classifiers built with meaningful counter examples.
>
> The goal of this type of anomaly detection is to provide some useful information where no information was previously attainable. However, if there are enough of the "rare" cases so that stratified sampling produces a training set with enough counterexamples for a standard classification model, then the classification may be a better solution.

Related Topics

- About Classification
  Classification is a machine learning technique that assigns items in a collection to target categories or classes.

6.1.2 Anomaly Detection for Time Series Data

With the growing number of sensors in the internet of things, the ability to identify anomalous events among potentially thousands of sensors is essential. For example, in the early detection of anomalies in business-critical enterprise computing servers and software systems, storage systems, and networks.

Enterprises require high anomaly detection accuracy, which implies lower false-alarm probabilities, lower missed-alarm probabilities, and lower overhead compute cost. The ability to distinguish between a real problem and sensor malfunction can significantly reduce costs in problem solution.

Building a model involves supplying historical, error-free operating data from, for example, monitored equipment. The resulting model is used to score new sensor data, also referred to as the *monitoring phase*, to estimate the expected sensor values.

6.2 Anomaly Detection Algorithms

For anomaly detection, Oracle Machine Learning for SQL has the following algorithms.

- Multivariate state Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT)
- One-Class Support Vector Machine (SVM)
- Expectation Maximization (EM) Anomaly

Anomaly detection is a form of classification. When you create a model using the MSET-SPRT and One-Class SVM and EM Anomaly algorithms, specify the classification machine learning technique. These algorithms do not use a target.

Related Topics

- Multivariate State Estimation Technique - Sequential Probability Ratio Test
  The Multivariate State Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT) algorithm monitors critical processes and detects subtle anomalies.
- One-Class SVM
  Support Vector Machine (SVM) as a one-class classifier is used for detecting anomalies.
- Expectation Maximization for Anomaly Detection
  An object is identified as an outlier in an EM Anomaly model if its anomaly probability is greater than 0.5.

7 Ranking

Ranking is a regression machine learning technique.

7.1 About Ranking

Ranking is a machine learning technique to rank items.

Ranking is useful for many applications in information retrieval such as e-commerce, social networks, recommendation systems, and so on. For example, a user searches for an article or an item to buy online. To build a recommendation system, it becomes important that similar articles or items of relevance appear to the user such that the user clicks or purchases the item. A simple regression model can predict the probability of a user to click an article or buy an item. However, it is more practical to use ranking technique and be able to order or rank the articles or items to maximize the chances of getting a click or purchase. The prioritization of the articles or the items influence the decision of the users.

The ranking technique directly ranks items by training a model to predict the ranking of one item over another item. In the training model, it is possible to have items, ranking one over the other by having a "score" for each item. Higher ranked items have higher scores and lower ranked items have lower scores. Using these scores, a model is built to predict which item ranks higher than the other.

7.2 Ranking Methods

Oracle Machine Learningsupports pairwise and listwise ranking methods through XGBoost.

For a training data set, in a number of sets, each set consists of objects and labels representing their ranking. A ranking function is constructed by minimizing a certain loss function on the training data. Using test data, the ranking function is applied to get a ranked list of objects. Ranking is enabled for XGBoost using the regression function. OML4SQL supports pairwise and listwise ranking methods through XGBoost.

Pairwise ranking: This approach regards a pair of objects as the learning instance. The pairs and lists are defined by supplying the same `case_id` value. Given a pair of objects, this approach gives an optimal ordering for that pair. Pairwise losses are defined by the order of the two objects. In OML4SQL, the algorithm uses LambdaMART to perform pairwise ranking with the goal of minimizing the average number of inversions in ranking.

Listwise ranking: This approach takes multiple lists of ranked objects as learning instance. The items in a list must have the same `case_id`. The algorithm uses LambdaMART to perform list-wise ranking.

> **See Also:**
>
> - "Ranking Measures and Loss Functions in Learning to Rank" a research paper presentation at https://www.researchgate.net/
> - *Oracle Database PL/SQL Packages and Types Reference* for a listing and explanation of the available model settings for XGBoost.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- About XGBoost
  Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.
- DBMS_DATA_MINING — Algorithm Settings: XGBoost

7.3 Ranking Algorithms

Ranking falls under the Regression function.

OML4SQL supports XGBoost algorithm for ranking.

Related Topics

- About XGBoost
  Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

8 Association

Learn how to discover association rules through association - an unsupervised machine learning technique.

8.1 About Association

Association is a Oracle Machine Learning for SQL function that discovers the probability of the co-occurrence of items in a collection.

The relationships between co-occurring items are expressed as **Association Rules**.

8.1.1 Association Rules

Identifies the pattern of association within the data.

The results of an association model are the rules that identify patterns of association within the data. Oracle Machine Learning for SQL does not support the scoring operation for association modeling.

Association rules can be applied as follows:

Support: How often do these items occur together in the data?
Confidence: How frequently the consequent occurs in transactions that contain the antecedent.
Value: How much business value is connected to item associations

8.1.2 Market-Basket Analysis

Association rules are often used to analyze sales transactions. For example, it is noted that customers who buy cereal at the grocery store often buy milk at the same time. In fact, association analysis find that 85% of the checkout sessions that include cereal also include milk. This relationship can be formulated as the following rule:

```
Cereal implies milk with 85% confidence
```

This application of association modeling is called **market-basket analysis**. It is valuable for direct marketing, sales promotions, and for discovering business trends. Market-basket analysis can also be used effectively for store layout, catalog design, and cross-sell.

8.1.3 Association Rules and eCommerce

Learn about application of association rules in other domains.

Association modeling has important applications in other domains as well. For example, in e-commerce applications, association rules may be used for Web page personalization. An association model might find that a user who visits pages A and B is 70% likely to also visit page C in the same session. Based on this rule, a dynamic link can be created for users who are likely to be interested in page C. The association rule is expressed as follows:

```
A and B imply C with 70% confidence
```

Related Topics

- Confidence
  The confidence of a rule indicates the probability of both the antecedent and the consequent appearing in the same transaction.

8.2 Transactional Data

Learn about transactional data, also known as market-basket data.

Unlike other machine learning functions, association is transaction-based. In transaction processing, a case includes a collection of items such as the contents of a market basket at the checkout counter. The collection of items in the transaction is an attribute of the transaction. Other attributes might be a timestamp or user ID associated with the transaction.

Transactional data, also known as **market-basket data**, is said to be in **multi-record case** format because a set of records (rows) constitute a case. For example, in the following figure, case 11 is made up of three rows while cases 12 and 13 are each made up of four rows.

Figure 8-1 Transactional Data

```
 case ID    attribute1   attribute2
    |           |            |
TRANS_ID    ITEM_ID      OPER_ID
---------   ---------    ---------
   11           B          m5203
   11           D          m5203
   11           E          m5203
   12           A          m5203
   12           B          m5203
   12           C          m5203
   12           E          m5203
   13           B          q5597
   13           C          q5597
   13           D          q5597
   13           E          q5597
```

Non transactional data is said to be in a **single-record case** format because a single record (row) constitutes a case. In Oracle Machine Learning for SQL, association models can be built using either transactional or non transactional or two-dimensional data formats. If the data is non transactional, it is possible to transform to a nested column to make it transactional before association machine learning activities can be performed. Transactional format is the usual format but, the association rules model does accept two-dimensional input format. For non transactional input format, each distinct combination of the content in all columns other than the case ID column is treated as a unique item.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*
- Data Preparation for Apriori

8.3 Association Algorithm

Oracle Machine Learning for SQL uses the Apriori algorithm to calculate association rules for items in frequent itemsets.

Related Topics

- About Apriori
  Learn how to find associations involving rare events in a large number of items using Apriori.

9
Feature Selection

Learn how to perform feature selection and attribute importance.

Oracle Machine Learning for SQL supports attribute importance as a supervised and unsurpervised machine learning technique .

9.1 Finding the Attributes

Find the attributes by using preprocessing steps to reduce the effect of noise, correlation, and high-dimensionality.

Sometimes too much information can reduce the effectiveness of OML4SQL. Some of the columns of data attributes assembled for building and testing a model in a supervised learning do not contribute meaningful information to the model. Some actually detract from the quality and accuracy of the model.

For example, you want to collect a great deal of data about a given population because you want to predict the likelihood of a certain illness within this group. Some of this information, perhaps much of it, has little or no effect on susceptibility to the illness. It is possible that attributes such as the number of cars per household do not have effect whatsoever.

Irrelevant attributes add noise to the data and can affect model accuracy. Noise increases the size of the model and the time and system resources needed for model building and scoring.

Data sets with many attributes can contain groups of attributes that are correlated. These attributes actually measure the same underlying feature. Their presence together in the build data can skew the patterns found by algorithm and affect the accuracy of the model.

Wide data (many attributes) typically results in more processing by machine learning algorithms. Model attributes are the dimensions of the processing space used by the algorithm. The higher the dimensionality of the processing space, the higher the computation cost involved in algorithmic processing.

To minimize the effects of noise, correlation, and high dimensionality, some form of dimension reduction is often a desirable preprocessing step. Feature selection involves identifying those attributes that are most predictive and selecting among those to provide the algorithm for model building. By removing attributes that add little or no value to a model has these benefits - potentially increasing model accuracy while reducing compute time since fewer attributes need to be processed. Informative and representative samples are best suited in feature selection. Sometimes you can represent the variables that are important than to represent the linear combination of variables. You can single-out and measure the "importance" of a column or a row in a data matrix in an unsupervised manner (a low-rank matrix decomposition).

Feature selection optimization is performed in the Decision Tree algorithm and within Naive Bayes as an algorithm behavior. The Generalized Linear Model (GLM) algorithm can be configured to perform feature selection through model setting.

9.2 About Feature Selection and Attribute Importance

Finding the most significant predictors is the goal of some machine learning projects. For example, a model might seek to find the principal characteristics of clients who pose a high credit risk.

Oracle Machine Learning for SQL supports the **attribute importance** machine learning technique, which ranks attributes according to their importance. Attribute importance does not actually select the features, but ranks them as to their relevance to predicting the result. It is up to the user to review the ranked features and create a data set to include the desired features.

Feature selection is useful as a preprocessing step to improve computational efficiency in predictive modeling.

9.2.1 Attribute Importance and Scoring

The results of attribute importance are the attributes of the build data ranked according to their influence.

The ranking and the measure of importance can be used in selecting training data for classification and regression models. Also, used for selecting data for unsupervised algorithm like CUR matrix decomposition. Oracle Machine Learning for SQL does not support the scoring operation for attribute importance.

9.3 Algorithms for Attribute Importance

Understand the algorithms used for attribute importance.

Oracle Machine Learning for SQL supports the following algorithms for attribute importance:

- Minimum Description Length
- CUR Matrix Decomposition

Related Topics

- About CUR Matrix Decomposition
  CUR Matrix Decomposition is a low-rank matrix decomposition algorithm that is explicitly expressed in a small number of actual columns and/or actual rows of data matrix.
- About MDL
  Minimum Description Length (MDL) is an information theoretic model selection principle.

10 Embedding

Explore embedding as a machine learning technique that transforms data in numeric dimensions that are represented as vectors to enable content similarity search and other applications.

10.1 About Vector Embeddings

Transformer models, also known as embedding models, are used to convert various types of data, such as words, sentences, documents, images, and more, into numerical vectors that capture their semantics.

These vectors are represented as points in a multidimensional space, where the proximity of points reflects the semantic similarity of the data they represent. Put differently, vector embeddings are a way of representing various types of data, like text, images, videos, or music, as points in a multidimensional space. The locations of these points and their proximity to others are semantically meaningful. This transformation enables machine learning algorithms to process and analyze data more effectively, and compute various distance metrics to find similar content. Creating vector embeddings involves training machine learning models, often neural networks, on large data sets to learn patterns and relationships within the data. This process transforms the data into numerical vectors, each uniquely representing a data point in a high-dimensional vector space. Applications of vector embeddings span a wide range of fields, particularly in natural language processing (NLP), search engines, and recommendation systems to name a few.

10.2 Pretrained Models for Generating Embeddings

Many pretrained models exist that generate embeddings for various data types, such as words, text sentences, images, and so on. These pretrained models often require pre-processing or post-processing operations or both.

As an example, most models for creating sentence embeddings from text require a pre-processing step called tokenization. **Tokenization** is a process to convert a sequence of text into smaller parts, called tokens. The embedding model then processes the tokens as input. Further post-processing might also be necessary for the output of these pretrained sentence transformers. One such post-processing operation is pooling. **Pooling** in text embeddings is a technique used to aggregate and reduce the dimensionality of individual word or token embeddings within a text sequence. This process involves combining the features of multiple embeddings to form a single, fixed-size representation of the entire text. For example, pooling methods can be employed to perform aggregation functions such as mean, max, or others. Another post-processing operation is normalization. **Normalization** in text embeddings is a process that adjusts the individual embeddings to have a uniform scale or distribution. This step involves transforming the embeddings so that they conform to a specific structure, often aiming to have a consistent length or scale across the data set.

Therefore, you need to use pretrained models that are augmented with pre-processing and post-processing operations to generate embeddings. This document illustrates examples that use the `my_embedding_model.onnx` model as an augmented ONNX format model. If you want to download and convert a pretrained model to an ONNX format model and augment the model with pre-processing and post-processing steps, see Import ONNX Models and Generate Embeddings.

10.3 Data Types for ONNX Embedding Models

ONNX defines its own data types. When you import ONNX models into Oracle Database, their data types are automatically mapped to SQL data types.

10.3.1 Attribute Data Type for ONNX Embedding Models

For a text embedding model, the input is a string. Therefore, the supported data type are `VARCHAR2`, `CLOB`, `NVARCHAR2`, and `NCLOB`. This means that there is a limit on the size of input strings to 4000 bytes (32767 bytes if the maximum string size is set to extended).

The `USER_MINING_MODEL_ATTRIBUTES` view reports the SQL data types for the input of a model.

```
USER_MINING_MODEL_ATTRIBUTESVARCHAR2

SELECT model_name, attribute_name, attribute_type, data_type,
vector_info
FROM user_mining_model_attributes
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;


MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE
DATA_TYPE  VECTOR_INFO
-------------------- -------------------- --------------------
---------- ---------------
DOC_MODEL                INPUT_STRING         TEXT
VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR
VECTOR     VECTOR(128,FLOA

   T32)
```

10.3.2 Target Data Type for ONNX Embedding Models

The output of a text embedding model is an embedding vector. Therefore, the target data type is `VECTOR`. Use the `VECTOR_EMBEDDING` SQL scoring function to generate vectors from an embedding model.

For more detail on `VECTOR` data type, see Create Tables Using the VECTOR Data Type. To learn more about `VECTOR_EMBEDDING` SQL operator, see *Oracle Database SQL Language Reference*.

10.4 Examples: Static Data Dictionary Views

You can use the Oracle Machine Learning static data dictionary views to view information such as available models, attributes of an ONNX embedding model and others. Values to support ONNX embedding models have been added.

Database administrator (DBA) and USER versions of the views are also available.

This section lists the examples of the impacted data dictionary views of ONNX embedding model.

10.4.1 Example: ALL_MINING_MODEL_ATTRIBUTES

You, as a current user, can view the attributes of a machine learning model by querying the `ALL_MINING_MODEL_ATTRIBUTES` view.

Here is an example of the model attributes of an embedding model. The name of the ONNX embedding model is `DOC_MODEL`:

```
SELECT model_name, attribute_name, attribute_type, data_type, vector_info
FROM user_mining_model_attributes
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;
```

The output is as follows:

```
MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE       DATA_TYPE
VECTOR_INFO
-------------------- -------------------- -------------------- ----------
---------------
DOC_MODEL                INPUT_STRING         TEXT                 VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR
VECTOR     VECTOR(128,FLOA

T32)
```

> ✎ **See Also:**
>
> ALL_MINING_MODEL_ATTRIBUTES in *Oracle Database Reference*

10.4.2 Example: ALL_MINING_MODELS

You can check machine learning models available to you as a current user by querying the `ALL_MINING_MODELS` view.

Here is an example of model details of an embedding model. The name of the ONNX embedding model is *DOC_MODEL*:

```
SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
ALGORITHM_TYPE, MODEL_SIZE
FROM user_mining_models
WHERE model_name = 'DOC_MODEL'
ORDER BY MODEL_NAME;
```

The output is as follows:

```
MODEL_NAME           MINING_FUNCTION
ALGORITHM            ALGORITHM_ MODEL_SIZE
-------------------- ------------------------------
-------------------- ---------- ----------
DOC_MODEL                EMBEDDING
ONNX                 NATIVE       17762137
```

See Also:

ALL_MINING_MODELS in *Oracle Database Reference*

10.5 Scoring: Generate Vector Embeddings

After importing the ONNX embedding model into the Database, you can generate embedding vectors using the `VECTOR_EMBEDDING` SQL scoring function.

The `VECTOR_EMBEDDING` SQL scoring function returns `VECTOR(dimension, type)`. The embedding models define the number of dimensions of the output vector of the `VECTOR_EMBEDDING` operator. To learn about the `VECTOR_EMBEDDING` SQL scoring operator, see VECTOR_EMBEDDING.

Example

The following example generates vector embeddings with "hello" as the input, utilizing the pretrained ONNX format model `my_embedding_model.onnx` imported into the Database. For complete example, see Import ONNX Models and Generate Embeddings.

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;
-----------------------------------------------------------------------
---------
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.
69305634E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13
885899E-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.158
```

```
8001E-002,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.6001057
6E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-
001,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E
-002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

Note:

You can define the outputs explicitly in the metadata or implicitly. The system assumes a single output for a model if you don't specify the output in the metadata.

If a scoring function does not comply as per the description in Supported SQL Scoring Functions, you will receive an ORA-40290 error when performing the scoring operation on your data. Additionally, any unsupported scoring functions will raise the ORA-40290 error.

See Also:

A complete list of SQL scoring functions supported for ONNX models, in *Oracle Machine Learning for SQL User’s Guide*.

10.5.1 Treatment of Missing Data During Scoring

ONNX does not support representation for non-existent values; that is, there is no equivalent to `NULL` for SQL.

Further, if the input values are not specified, then the ONNX embedding models fail to run.

- Absent attribute: If fewer attributes are used for scoring than were specified during model import (input), then you receive an error when you perform scoring. That is, if at least one of the input value is not specified in the `USING` clause of a scoring operator with ONNX model, then the query will not compile.
- `NULL` attribute: If any of the attributes has a `NULL` value, then the scoring operator does not perform inference of the model with the ONNX Runtime and returns a `NULL` result immediately. If you want to change this behavior, then provide an appropriate replacement to the NULL value, either by using an `NVL` expression as input attribute (for example, `NVL(input_attribute, default_value) AS input_attribute);`) or by specifying a default value for this input attribute using the JSON metadata when importing the model.

10.6 Import ONNX Models and Generate Embeddings

Learn to import a pretrained embedding model that is in ONNX format and generate vector embeddings.

Follow the steps below to import a pertained ONNX formatted embedding model into the Oracle Database.

Prepare Your Data Dump Directory

Prepare your data dump directory and provide the necessary access and privileges to `dmuser`.

1. Choose from:
    a. If you already have a pretrained ONNX embedding model, store it in your working folder.
    b. If you do not have pretrained embedding model in ONNX format, perform the steps listed in Convert Pretrained Models to ONNX Format.
2. Login to SQL*Plus as `SYSDBA` in your PDB.

    ```
    CONN sys/<password>@pdb as sysdba;
    ```

3. Grant the `DB_DEVELOPER_ROLE` to `dmuser`.

    ```
    GRANT DB_DEVELOPER_ROLE TO dmuser identified by <password>;
    ```

4. Grant `CREATE MINING MODEL` privilege to `dmuser`.

    ```
    GRANT create mining model TO dmuser;
    ```

5. Set your working folder as the data dump directory (`DM_DUMP`) to load the ONNX embedding model.

    ```
    CREATE OR REPLACE DIRECTORY DM_DUMP as '<work directory path>';
    ```

6. Grant `READ` permissions on the `DM_DUMP` directory to `dmuser`.

    ```
    GRANT READ ON DIRECTORY dm_dump TO dmuser;
    ```

7. Grant `WRITE` permissions on the `DM_DUMP` directory to `dmuser`.

    ```
    GRANT WRITE ON DIRECTORY dm_dump TO dmuser;
    ```

8. Drop the model if it already exits.

    ```
    exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force
    => true);
    ```

Import ONNX Model Into the Database

You created a data dump directory and now you load the ONNX model into the Database. Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to load the model. The `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure facilitates the process of importing ONNX format model into the Oracle Database. In this example, the procedure loads an ONNX model file, named `my_embedding_model.onnx` from the `DM_DUMP` directory, into the Database as `doc_model`, specifying its use for embedding tasks.

1. Connect as `dmuser`.

   ```
   CONN dmuser/<password>@<pdbname>;
   ```

2. Load the ONNX model into the Database.

   If the ONNX model to be imported already includes an output tensor named `embeddingOutput` and an input string tensor named `data`, JSON metadata is unnecessary. Embedding models converted from OML4Py follow this convention and can be imported without the JSON metadata.

   ```
   EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
     'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model');
   ```

   Alternately, you can load the ONNX embedding model by specifying the JSON metadata.

```
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
     'DM_DUMP',
     'my_embedding_model.onnx',
     'doc_model',
     JSON('{"function" : "embedding", "embeddingOutput" : "embedding", "input": {"input":
["DATA"]}}'));
```

The procedure `LOAD_ONNX_MODEL` declares these parameters:

- `DM_DUMP`: specifies the directory name of the data dump.

  > **Note:**
  >
  > Ensure that the `DM_DUMP` directory is defined.

- `my_embedding_model`: is a `VARCHAR2` type parameter that specifies the name of the ONNX model.
- `doc_model`: This parameter is a user-specified name under which the model is stored in the Oracle Database.
- The JSON metadata associated with the ONNX model is declared as:

  `"function" : "embedding"`: Indicates the function name for text embedding model.

  `"embeddingOutput" : "embedding"`: Specifies the output variable which contains the embedding results.

- `"input": {"input": ["DATA"]}`: Specifies a JSON object (`"input"`) that describes the input expected by the model. It specifies that there is an input named `"input"`, and its value should be an array with one element, `"DATA"`. This indicates that the model expects a single string input to generate embeddings.

See LOAD_ONNX_MODEL Procedure to learn about the PL/SQL procedure.

Query Model Statistics

You can view model attributes and learn about the model by querying machine learning dictionary views and model detail views.

> **Note:**
>
> *DOC_MODEL* is the user-specified name of the embedding text model.

1. Query `USER_MINING_MODEL_ATTRIBUTES` view.

   ```
   SELECT model_name, attribute_name, attribute_type, data_type,
   vector_info
   FROM user_mining_model_attributes
   WHERE model_name = 'DOC_MODEL'
   ORDER BY ATTRIBUTE_NAME;
   ```

   To learn about `USER_MINING_MODEL_ATTRIBUTES` view, see USER_MINING_MODEL_ATTRIBUTES.

2. Query `USER_MINING_MODELS` view.

   ```
   SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
   ALGORITHM_TYPE, MODEL_SIZE
   FROM user_mining_models
   WHERE model_name = 'DOC_MODEL'
   ORDER BY MODEL_NAME;
   ```

   To learn about `USER_MINING_MODELS` view, see USER_MINING_MODELS.

3. Check model statistics by viewing the model detail views. Query the `DM$VMDOC_MODEL` view.

   ```
   SELECT * FROM DM$VMDOC_MODEL ORDER BY NAME;
   ```

   To learn about model details views for ONNX embedding models, see Model Details Views for ONNX Models.

4. Query the `DM$VPDOC_MODEL` model detail view.

   ```
   SELECT * FROM DM$VPDOC_MODEL ORDER BY NAME;
   ```

5. Query the `DM$VJDOC_MODEL` model detail view.

```
SELECT * FROM DM$VJDOC_MODEL;
```

Generate Embeddings

Apply the model and generate vector embeddings for your input. Here, the input is *hello*.

Generate vector embeddings using the `VECTOR_EMBEDDING` function.

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS
embedding;
```

To learn about the `VECTOR_EMBEDDING` SQL function, see VECTOR_EMBEDDING. You can use the `UTL_TO_EMBEDDING` function in the `DBMS_VECTOR_CHAIN` PL/SQL package to generate vector embeddings generically through REST endpoints. To explore these functions, see the example Convert Text String to Embedding.

Example: Importing a Pretrained ONNX Model to Oracle Database

The following presents a comprehensive step-by-step example of importing ONNX embedding and generating vector embeddings.

```
conn sys/<password>@pdb as sysdba
grant db_developer_role to dmuser identified by dmuser;
grant create mining model to dmuser;

create or replace directory DM_DUMP as '<work directory path>';
grant read on directory dm_dump to dmuser;
grant write on directory dm_dump to dmuser;
>conn dmuser/<password>@<pdbname>;

-- Drop the model if it exits
exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force => true);

-- Load Model
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" : "embedding"}'));
/

--check the attributes view
set linesize 120
col model_name format a20
col algorithm_name format a20
col algorithm format a20
col attribute_name format a20
col attribute_type format a20
col data_type format a20

SQL> SELECT model_name, attribute_name, attribute_type, data_type,
vector_info
```

```
FROM user_mining_model_attributes
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;


OUTPUT:

MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE
DATA_TYPE  VECTOR_INFO
-------------------- -------------------- --------------------
---------- ---------------
DOC_MODEL                INPUT_STRING         TEXT
VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR
VECTOR     VECTOR(128,FLOA

   T32)



SQL> SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
ALGORITHM_TYPE, MODEL_SIZE
FROM user_mining_models
WHERE model_name = 'DOC_MODEL'
ORDER BY MODEL_NAME;

OUTPUT:
MODEL_NAME           MINING_FUNCTION
ALGORITHM            ALGORITHM_ MODEL_SIZE
-------------------- ------------------------------
-------------------- ---------- ----------
DOC_MODEL                EMBEDDING
ONNX                 NATIVE       17762137



SQL> select * from DM$VMDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                     VALUE
----------------------------------------
----------------------------------------
Graph Description                        Graph combining g_8_torch_jit
and torch_
                                         jit
                                         g_8_torch_jit



                                         torch_jit



Graph Name                               g_8_torch_jit_torch_jit
Input[0]                                 input:string[1]
Output[0]                                embedding:float32[?,128]
```

```
Producer Name                           onnx.compose.merge_models
Version                                 1

6 rows selected.


SQL> select * from DM$VPDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                    VALUE
----------------------------------------
----------------------------------------
batching                                False
embeddingOutput                         embedding


SQL> select * from DM$VJDOC_MODEL;

OUTPUT:
METADATA
-----------------------------------------------------------------------------
---
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
["DATA"]}}



--apply the model
SQL> SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as data)) AS 
embedding;

-----------------------------------------------------------------------------
---
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.693056
34E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13885899
E-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-
002,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.6001057
6E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-
001,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E
-002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

Oracle AI Vector Search SQL Scenario

To learn how you can chunk *database-concepts23ai.pdf* and *oracle-ai-vector-search-users-guide.pdf*, generate vector embeddings, and perform similarity search using vector indexes, see Quick Start SQL.

10.6.1 Alternate Method to Import ONNX Models

Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure to import the model and declare the input name. The following procedure uses a PL/SQL helper block that facilitates the process of importing ONNX format model into the Oracle Database. The function reads the model file from the server's file system and imports it into the Database.

Perform the following steps to import ONNX model into the Database using `DBMS_DATA_MINING` PL/SQL package.

- Connect as `dmuser`.

  ```
  CONN dmuser/<password>@<pdbname>;
  ```

- Run the following helper PL/SQL block:

  ```
  DECLARE
      m_blob BLOB default empty_blob();
      m_src_loc BFILE ;
      BEGIN
      DBMS_LOB.createtemporary (m_blob, FALSE);
      m_src_loc := BFILENAME('DM_DUMP', 'my_embedding_model.onnx');
      DBMS_LOB.fileopen (m_src_loc, DBMS_LOB.file_readonly);
      DBMS_LOB.loadfromfile (m_blob, m_src_loc, DBMS_LOB.getlength
  (m_src_loc));
      DBMS_LOB.CLOSE(m_src_loc);
      DBMS_DATA_MINING.import_onnx_model ('doc_model', m_blob,
  JSON('{"function" : "embedding", "embeddingOutput" : "embedding",
  "input": {"input": ["DATA"]}}'));
      DBMS_LOB.freetemporary (m_blob);
      END;
      /
  ```

  The code sets up a `BLOB` object and a `BFILE` locator, creates a temporary `BLOB` for storing the `my_embedding_model.onnx` file from the `DM_DUMP` directory, and reads its contents into the `BLOB`. It then closes the file and uses the content to import an ONNX model into the database with specified metadata, before releasing the temporary `BLOB` resources.

The schema of the `IMPORT_ONNX_MODEL` procedure is as follows: `DBMS_DATA_MINING.IMPORT_ONNX_MODEL(model_data, model_name, metadata)`. This procedure loads `IMPORT_ONNX_MODEL` from the `DBMS_DATA_MINING` package to import the ONNX model into the Database using the name provided in `model_name`, the BLOB content in `m_blob`, and the associated `metadata`.

- `doc_model`: This parameter is a user-specified name under which the imported model is stored in the Oracle Database.
- `m_blob`: This is a model data in `BLOB` that holds the ONNX representation of the model.
- `"function" : "embedding"`: Indicates the function name for text embedding model.

- `"embeddingOutput" : "embedding"`: Specifies the output variable which contains the embedding results.
- `"input": {"input": ["DATA"]}`: Specifies a JSON object (`"input"`) that describes the input expected by the model. It specifies that there is an input named `"input"`, and its value should be an array with one element, `"DATA"`. This indicates that the model expects a single string input to generate embeddings.

Alternately, the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure can also accept a `BLOB` argument representing an ONNX file stored and loaded from OCI Object Storage. The following is an example to load an ONNX model stored in an OCI Object Storage.

```
DECLARE
  model_source BLOB := NULL;
BEGIN
  -- get BLOB holding onnx model
  model_source := DBMS_CLOUD.GET_OBJECT(
    credential_name => 'myCredential',
    object_uri => 'https://objectstorage.us-phoenix -1.oraclecloud.com/' ||
      'n/namespace -string/b/bucketname/o/myONNXmodel.onnx');

  DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
    "myonnxmodel",
    model_source,
    JSON('{ function : "embedding" })
  );
END;
/
```

This PL/SQL block starts by initializing a `model_source` variable as a `BLOB` type, initially set to NULL. It then retrieves an ONNX model from Oracle Cloud Object Storage using the `DBMS_CLOUD.GET_OBJECT` procedure, specifying the credentials `(OBJ_STORE_CRED`) and the URI of the model. The ONNX model resides in a specific bucket named `bucketname` in this case, and is accessible through the provided URL. Then, the script loads the ONNX model into the `model_source` BLOB. The `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure then imports this model into the Oracle Database as `myonnxmodel`. During the import, a JSON metadata specifies the model's function as `embedding`, for embedding operations.

See IMPORT_ONNX_MODEL Procedure and GET_OBJECT Procedure and Function to learn about the PL/SQL procedure.

Example: Importing a Pretrained ONNX Model to Oracle Database

The following presents a comprehensive step-by-step example of importing ONNX embedding and generating vector embeddings.

```
conn sys/<password>@pdb as sysdba
grant db_developer_role to dmuser identified by dmuser;
grant create mining model to dmuser;

create or replace directory DM_DUMP as '<work directory path>';
grant read on directory dm_dump to dmuser;
grant write on directory dm_dump to dmuser;
>conn dmuser/<password>@<pdbname>;
```

```
-- Drop the model if it exits
exec DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'doc_model', force =>
true);

-- Load Model
EXECUTE DBMS_VECTOR.LOAD_ONNX_MODEL(
    'DM_DUMP',
    'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding", "embeddingOutput" :
"embedding"}'));
/
--Alternately, load the model
EXECUTE DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
       'my_embedding_model.onnx',
    'doc_model',
    JSON('{"function" : "embedding",
    "embeddingOutput" : "embedding",
    "input": {"input": ["DATA"]}}')
    );

--check the attributes view
set linesize 120
col model_name format a20
col algorithm_name format a20
col algorithm format a20
col attribute_name format a20
col attribute_type format a20
col data_type format a20

SQL> SELECT model_name, attribute_name, attribute_type, data_type,
vector_info
FROM user_mining_model_attributes
WHERE model_name = 'DOC_MODEL'
ORDER BY ATTRIBUTE_NAME;


OUTPUT:

MODEL_NAME           ATTRIBUTE_NAME       ATTRIBUTE_TYPE
DATA_TYPE  VECTOR_INFO
-------------------- -------------------- --------------------
---------- ---------------
DOC_MODEL                INPUT_STRING         TEXT
VARCHAR2
DOC_MODEL                ORA$ONNXTARGET       VECTOR
VECTOR     VECTOR(128,FLOA

   T32)



SQL> SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM,
ALGORITHM_TYPE, MODEL_SIZE
```

```
FROM user_mining_models
WHERE model_name = 'DOC_MODEL'
ORDER BY MODEL_NAME;

OUTPUT:
MODEL_NAME           MINING_FUNCTION                ALGORITHM
ALGORITHM_ MODEL_SIZE
-------------------- ------------------------------ --------------------
---------- ----------
DOC_MODEL                EMBEDDING                      ONNX
NATIVE       17762137



SQL> select * from DM$VMDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                    VALUE
----------------------------------------
----------------------------------------
Graph Description                       Graph combining g_8_torch_jit and
torch_
                                        jit
                                        g_8_torch_jit



                                        torch_jit



Graph Name                              g_8_torch_jit_torch_jit
Input[0]                                input:string[1]
Output[0]                               embedding:float32[?,128]
Producer Name                           onnx.compose.merge_models
Version                                 1

6 rows selected.


SQL> select * from DM$VPDOC_MODEL ORDER BY NAME;

OUTPUT:
NAME                                    VALUE
----------------------------------------
----------------------------------------
batching                                False
embeddingOutput                         embedding


SQL> select * from DM$VJDOC_MODEL;

OUTPUT:
METADATA
-----------------------------------------------------------------------------
---
```

```
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
["DATA"]}}



--apply the model
SQL> SELECT TO_VECTOR(VECTOR_EMBEDDING(doc_model USING 'hello' as
data)) AS embedding;

------------------------------------------------------------------------
---------
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.
69305634E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13
885899E-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.158
8001E-002,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.6
0010576E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.1685
1285E-001,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.424
07483E-002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```

11

Feature Extraction

Learn how to perform attribute reduction using feature extraction as an unsupervised function.

Oracle Machine Learning for SQL supports feature extraction as an unsupervised machine learning function.

11.1 About Feature Extraction

Feature extraction is a dimensionality reduction technique. Unlike feature selection, which selects and retains the most significant attributes, feature extraction actually transforms the attributes. The transformed attributes, or **features**, are linear combinations of the original attributes.

The feature extraction technique results in a much smaller and richer set of attributes. The maximum number of features can be user-specified or determined by the algorithm. By default, the algorithm determines it.

Models built on extracted features can be of higher quality, because fewer and more meaningful attributes describe the data.

Feature extraction projects a data set with higher dimensionality onto a smaller number of dimensions. As such it is useful for data visualization, since a complex data set can be effectively visualized when it is reduced to two or three dimensions.

Some applications of feature extraction are latent semantic analysis, data compression, data decomposition and projection, and pattern recognition. Feature extraction can also be used to enhance the speed and effectiveness of machine learning algorithms.

Feature extraction can be used to extract the themes of a document collection, where documents are represented by a set of key words and their frequencies. Each theme (feature) is represented by a combination of keywords. The documents in the collection can then be expressed in terms of the discovered themes.

11.1.1 Feature Extraction and Scoring

Oracle Machine Learning for SQL supports the scoring operation for feature extraction. As an unsupervised machine learning technique, feature extraction does not involve a target. When applied, a feature extraction model transforms the input into a set of features.

11.2 Algorithms for Feature Extraction

Understand the algorithms used for feature extraction.

OML4SQL supports these feature extraction algorithms:

- **Explicit Semantic Analysis** (ESA).
- **Non-Negative Matrix Factorization** (NMF).
- **Singular Value Decomposition** (SVD) and **Principal Component Analysis** (PCA).

> **Note:**
>
> OML4SQL uses NMF as the default feature extraction algorithm.

Related Topics

- About Explicit Semantic Analysis
  In Oracle Database 12*c* Release 2, Explicit Semantic Analysis (ESA) was introduced as an unsupervised algorithm for feature extraction. Starting from Oracle Database 18c, ESA is enhanced as a supervised algorithm for classification.
- About NMF
  Non-Negative Matrix Factorization is useful when there are many attributes and the attributes are ambiguous or have weak predictability. By combining attributes, NMF can produce meaningful patterns, topics, or themes. NMF is a feature extraction algorithm.
- About Singular Value Decomposition
  SVD and the closely-related PCA are well established feature extraction methods that have a wide range of applications. Oracle Machine Learning for SQL implements Singular Value Decomposition (SVD) as a feature extraction algorithm and Principal Component Analysis (PCA) as a special scoring method for SVD models.
- PCA scoring
  Learn about configuring Singular Value Decomposition (SVD) to perform Principal Component Analysis (PCA) projections.

12 Row Importance

Row importance is an unsupervised machine learning technique that can be applied to data as a preprocessing step prior to model building using other mining functions and algorithms.

12.1 About Row Importance

Row importance captures the influence of the rows or cases in a data set.

Row importance technique is used in dimensionality reduction of large data sets. Row importance identifies the most influential rows of the data matrix. The rows with high importance are ranked by their importance scores. The "importance" of a row is determined by high statistical leverage scores. In CUR matrix decomposition, row importance is often combined with column (attribute) importance. Row importance can serve as a data preprocessing step prior to model building using regression, classification, and clustering.

Related Topics

- About CUR Matrix Decomposition
  CUR Matrix Decomposition is a low-rank matrix decomposition algorithm that is explicitly expressed in a small number of actual columns and/or actual rows of data matrix.
- Statistical Leverage Score
  Leverage scores are statistics that determine which column (or rows) are most representative with respect to a rank subspace of a matrix. The statistical leverage scores represent the column (or attribute) and row importance.
- CUR Matrix Decomposition Algorithm Configuration
  Configure the CUR Matrix Decomposition algorithm setting to build your model.

12.2 Selecting Important Rows

The rows with high importance are ranked by their importance scores. The "importance" of a row is determined by high statistical leverage scores.

Row importance, that is, rows with high leverage scores are reported as names (as `case_id`), scores (as importance), and ranks (by importance).

12.3 Row Importance Algorithms

Oracle Machine Learning for SQL supports CUR matrix decomposition algorithm for row and column (attribute) importance.

Popular algorithms for dimensionality reduction are Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and CUR Matrix Decomposition. All these algorithms apply low-rank matrix decomposition.

In CUR matrix decomposition, the attributes include 2-Dimensional numerical columns, levels of exploded 2D categorical columns, and attribute name or subname or value pairs for nested columns. To arrive at row importance or selection, the algorithm computes singular vectors, calculates leverage scores, and then selects rows. Row importance is performed when users specify `CURS_ROW_IMP_ENABLE` for the `CURS_ROW_IMPORTANCE` parameter in the settings table and the `case_id` column is present. Unless users explicitly specify, row importance is not performed.

Related Topics

- About Singular Value Decomposition
  SVD and the closely-related PCA are well established feature extraction methods that have a wide range of applications. Oracle Machine Learning for SQL implements Singular Value Decomposition (SVD) as a feature extraction algorithm and Principal Component Analysis (PCA) as a special scoring method for SVD models.
- About CUR Matrix Decomposition
  CUR Matrix Decomposition is a low-rank matrix decomposition algorithm that is explicitly expressed in a small number of actual columns and/or actual rows of data matrix.

13 Time Series

Learn about time series as an Oracle Machine Learning for SQL regression function.

13.1 About Time Series

Time series is a machine learning technique that forecasts target value based solely on a known history of target values. It is a specialized form of regression, known in the literature as auto-regressive modeling.

The input to time series analysis is a sequence of target values. A case id column specifies the order of the sequence. The case id can be of type `NUMBER` or a date type (date, datetime, timestamp with timezone, or timestamp with local timezone). Regardless of case id type, the user can request that the model include trend, seasonal effects or both in its forecast computation. When the case id is a date type, the user must specify a time interval (for example, month) over which the target values are to be aggregated, along with an aggregation procedure (for example, sum). Aggregation is performed by the algorithm prior to constructing the model.

The time series model provide estimates of the target value for each step of a time window that can include up to 30 steps beyond the historical data. Like other regression models, time series models compute various statistics that measure the goodness of fit to historical data.

Forecasting is a critical component of business and governmental decision making. It has applications at the strategic, tactical and operation level. The following are the applications of forecasting:

- Projecting return on investment, including growth and the strategic effect of innovations
- Addressing tactical issues such as projecting costs, inventory requirements and customer satisfaction
- Setting operational targets and predicting quality and conformance with standards

Related Topics

- About Regression
  Regression is an Oracle Machine Learning for SQL function that predicts numeric values along a continuum.

13.2 Choosing a Time Series Model

Selecting a model depends on recognizing the patterns in the time series data. Consider trend, seasonality, or both that affect the data.

Time series data may contain patterns that can affect predictive accuracy. For example, during a period of economic growth, there may be an upward trend in sales. Sales may increase in specific seasons (bathing suits in summer). To accommodate such series, it can be useful to choose a model that incorporates trend, seasonal effects, or both.

Trend can be difficult to estimate, when you must represent trend by a single constant. For example, if there is a grow rate of 10%, then after 7 steps, the value doubles. Local growth rates, appropriate to a few time steps can easily approach such levels, but thereafter drop. **Damped trend** models can more accurately represent such data, by reducing cumulative trend effects. Damped trend models can better represent variability in trend effects over the historical data. Damped trend models are a good choice when the data have significant, but variable trend.

Since modeling attempts to reduce error, how error is measured can affect model predictions. For example, data that exhibit a wide range of values may be better represented by error as fraction of level. An error of a few hundred feet in the measurement of the height of a mountain may be equivalent to an error of an inch or two in the measurement of the height of a child. Errors that are measured relative to value are called **multiplicative errors**. Errors that are the same across values are called **additive errors**. If there are multiplicative effects in the model, then the error type is multiplicative. If there are no explicit multiplicative effects, error type is left to user specification. The type need not be the same across individual effects. For example, trend can be additive while seasonality is multiplicative. This particular mixed type effect combination defines the popular Holt-Winters model.

> **Note:**
>
> Multiplicative error is not an appropriate choice for data that contain zeros or negative values. Thus, when the data contains such values, it is best not to choose a model with multiplicative effects or to set error type to be multiplicative.

13.3 Automated Model Search

If you do not specify a model type (`EXSM_MODEL`) the default behavior is for the algorithm to automatically determine the model type.

The ESM settings are listed in DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing. Time Series model search considers a variety of models and selects the best one. For seasonal models, the seasonality is automatically determined.

The following example displays a sample code snippet that you can use for creating a model that automatically selects the best ESM model. In this example, `EXSM_MODEL` setting is not defined thereby allowing the algorithm to select the best model.

```
BEGIN DBMS_DATA_MINING.DROP_MODEL('ESM_SALES_FORECAST_1');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN

    v_setlst('ALGO_NAME')            := 'ALGO_EXPONENTIAL_SMOOTHING';
    v_setlst('EXSM_INTERVAL')        := 'EXSM_INTERVAL_QTR';
    v_setlst('EXSM_PREDICTION_STEP') := '4';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME           => 'ESM_SALES_FORECAST_1',
        MINING_FUNCTION      => 'TIME_SERIES',
        DATA_QUERY           => 'select * from ESM_SH_DATA',
        SET_LIST             => v_setlst,
        CASE_ID_COLUMN_NAME => 'TIME_ID',
        TARGET_COLUMN_NAME  => 'AMOUNT_SOLD');
END;
/
```

13.4 Time Series Statistics

Learn to evaluate model quality by applying commonly used statistics.

As with other regression functions, there are commonly used statistics for evaluating the overall model quality. An expert user can also specify one of these figures of merit as criterion to optimize by the model build process. Choosing an optimization criterion is not required because model-specific defaults are available.

13.4.1 Conditional Log-Likelihood

Log-likelihood is a figure of merit often used as an optimization criterion for models that provide probability estimates for predictions which depend on the values of the model's parameters.

The model probability estimates for the actual values in the training data then yields an estimate of the likelihood of the parameter values. Parameter values that yield high probabilities for the observed target values have high likelihood, and therefore indicate a good model. The calculation of log-likelihood depends on the form of the model.

Conditional log-likelihood breaks the parameters into two groups. One group is assumed to be correct and the other is assumed the source of any errors. Conditional log-likelihood is the log-likelihood of the latter group conditioned on the former group. For example, Exponential Smoothing (ESM) models make an estimate of the initial model state. The conditional log-likelihood of an ESM model is conditional on that initial model state (assumed to be correct). The ESM conditional log-likelihood is as follows:

$$L^*(\theta, X_0) = n \ln\left(\sum_{t=1}^{n} e_t^2 / k^2(x_{t-1})\right) + 2\sum_{t=1}^{n} \ln|k(x_{t-1})|$$

where e_t is the error at time `t` and `k(x(t-1) )` is `1` for ESM models with additive errors and is the estimated level at the previous time step in models with multiplicative error.

13.4.2 Mean Square Error (MSE) and Other Error Measures

Another time series figure of merit, that can also be used as an optimization criterion, is Mean Square Error (MSE).

The mean square error is computed as:

$$MSE = \sum_{t=1}^{n} {e_t}^2 / n$$

where the error at time *t* is the difference between the actual and model one step ahead forecast value at time *t* for models with additive error and that difference divided by the one-step ahead forecast for models with multiplicative error.

> **Note:**
>
> These "forecasts" are for over periods already observed and part of the input time series.

Since time series models can forecast for each of multiple steps ahead, time series can measure the error associated with such forecasts. Average Mean Square Error (AMSE), another figure of merit, does exactly that. For each period in the input time series, it computes a multi-step forecast, computes the error of those forecasts and averages the errors. AMSE computes the individual errors exactly as MSE does taking cognizance of error type (additive or multiplicative). The number of steps, *k*, is determined by the user (default 3). The formula is as follows:

$$AMSE = \sum_{t=1}^{n} \left(\sum_{\mathrm{i}=0}^{\mathrm{k}-1} e_{t+i}^2 / \mathrm{k}\right) \bigg/ n$$

Other figure of merit relatives of MSE include the Residual Standard Error (RMSE), which is the square root of MSE, and the Mean Absolute Error (MAE) which is the average of the absolute value of the errors.

13.4.3 Irregular Time Series

Irregular time series are time series data where the time intervals between observed values are not equally spaced.

One common practice is for the time intervals between adjacent steps to be equally spaced. However, it is not always convenient or realistic to force such spacing on time series. Irregular time series do not make the assumption that time series are equally spaced, but instead use the case id's date and time values to compute the intervals between observed values. Models are constructed directly on the observed values with their observed spacing. Oracle time series analysis handles irregular time series.

13.4.4 Build and Apply

A new time series model is built when new data arrives.

Many of the Oracle Machine Learning for SQL functions have separate build and apply operations, because you can construct and potentially apply a model to many different sets of input data. However, time series input consists of the target value history only. Thus, there is only one set of appropriate input data. When new data arrive, good practice dictates that a new model be built. Since the model is only intended to be used once, the model statistics and forecasts are produced during model build and are available through the model views.

13.5 Time Series Algorithm

Oracle Machine Learning for SQL uses the Exponential Smoothing algorithm to forecast from time series data.

Related Topics

- About Exponential Smoothing
  Exponential smoothing is a forecasting method for time series data. It is a moving average method where exponentially decreasing weights are assigned to past observations.

Part III

Algorithms

Oracle Machine Learning for SQL supports the algorithms listed in Part III. Part III provides basic conceptual information about the algorithms. There is at least one algorithm for each of the machine learning techniques.

Part III contains these chapters:

Related Topics

- Machine Learning Techniques
  Part II provides basic conceptual information about machine learning techniques that the Oracle Machine Learning for SQL supports.

14
Apriori

Learn how to calculate association rules using the Apriori algorithm.

- About Apriori
- Association Rules and Frequent Itemsets
- Data Preparation for Apriori
- Calculating Association Rules
- Evaluating Association Rules

Related Topics

- Association
  Learn how to discover association rules through association - an unsupervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- Machine Learning Function Settings
- Automatic Data Preparation
- Model Detail Views for Association Rules
- OML4SQL Examples
- OML4R Association Rules Example
- OML4R GitHub Examples

14.1 About Apriori

Learn how to find associations involving rare events in a large number of items using Apriori.

An association machine learning problem can be decomposed into the following subproblems:

- Find all combinations of items in a set of transactions that occur with a specified minimum frequency. These combinations are called **frequent itemsets**.
- Calculate rules that express the probable co-occurrence of items within frequent itemsets.

Apriori calculates the probability of an item being present in a frequent itemset, given that another item or items is present.

Association rule machine learning is not recommended for finding associations involving rare events in problem domains with a large number of items. Apriori discovers patterns with frequencies above the minimum support threshold. Therefore, to find associations involving rare events, the algorithm must run with very low minimum support values. However, doing so potentially explodes the number of enumerated itemsets, especially in cases with a large number of items. This increases the execution time significantly. Classification or anomaly detection is more suitable for discovering rare events when the data has a high number of attributes.

The build process for Apriori supports parallel execution.

Related Topics

- Example: Calculating Rules from Frequent Itemsets
  Example to calculating rules from frequent itemsets.
- *Oracle Database VLDB and Partitioning Guide*

14.2 Association Rules and Frequent Itemsets

The Apriori algorithm calculates rules that express probabilistic relationships between items in frequent itemsets. For example, a rule derived from frequent itemsets containing A, B, and C might state that if A and B are included in a transaction, then C is likely to also be included.

An association rule states that an item or group of items implies the presence of another item with some probability. Unlike decision tree rules, which predict a target, association rules express correlation.

14.2.1 Antecedent and Consequent

Defines antecedent and consequent in an Apriori algorithm.

The IF component of an association rule is known as the **antecedent**. The THEN component is known as the **consequent**. The antecedent and the consequent are disjoint; they have no items in common.

Oracle Machine Learning for SQL supports association rules that have one or more items in the antecedent and a single item in the consequent.

14.2.2 Confidence

Rules have an associated confidence, which is the conditional probability that the consequent occurs given the occurrence of the antecedent. You can specify the minimum confidence for rules.

14.3 Data Preparation for Apriori

Association models are designed to use transactional data. In transactional data, there is a one-to-many relationship between the case identifier and the values for each case. Each case ID/value pair is specified in a separate record (row).

14.3.1 Native Transactional Data and Star Schemas

Learn about storage format of transactional data.

Transactional data may be stored in native transactional format, with a non-unique case ID column and a values column, or it may be stored in some other configuration, such as a star schema. If the data is not stored in native transactional format, it must be transformed to a nested column for processing by the Apriori algorithm.

Related Topics

- Transactional Data
  Learn about transactional data, also known as market-basket data.
- *Oracle Machine Learning for SQL User's Guide*

14.3.2 Items and Collections

In transactional data, a collection of items is associated with each case. The collection theoretically includes all possible members of the collection. For example, all products can theoretically be purchased in a single market-basket transaction. However, in actuality, only a tiny subset of all possible items are present in a given transaction; the items in the market-basket represent only a small fraction of the items available for sale in the store.

14.3.3 Sparse Data

Understand how sparse data is used in the Apriori algorithm.

Missing items in a collection indicate **sparsity**. Missing items may be present with a null value, or they may be missing.

Nulls in transactional data are assumed to represent values that are known but not present in the transaction. For example, three items out of hundreds of possible items might be purchased in a single transaction. The items that were not purchased are known but not present in the transaction.

Oracle Machine Learning for SQL assumes sparsity in transactional data. The Apriori algorithm is optimized for processing sparse data.

> **Note:**
>
> Apriori is not affected by Automatic Data Preparation.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

14.3.4 Improved Sampling

Association rules (AR) can use a good sample size with performance guarantee, based on the work of Riondato and Upfal.

The AR algorithm computes the sample size by the following inputs:

- *d*-index of the dataset
- Absolute error ε
- Confidence level γ

d-index is defined as the maximum integer *d* such that the dataset contains at least *d* transactions of length *d* at the minimum. It is the upper bound of Vapnik-Chervonenkis (VC) dimension. The AR algorithm computes *d*-index of the dataset by scanning the length of all transactions in the dataset.

Users specify absolute error ε and confidence level γ parameters. A large *d*-index, small AR support, small ε or large γ can cause a large sample size. The sample size theoretically guarantees that the absolute error of both the support and confidence of the approximated AR (from sampling) is less than ε compared to the exact AR with probability (or confidence level) at least γ. In this document this sample size is called AR-specific sample size.

14.3.4.1 Sampling Implementation

The sample size is only computed when users turn on the sampling (`ODMS_SAMPLING` is set as `ODMS_SAMPLING_ENABLE`) and do not specify the sample size (`ODMS_SAMPLE_SIZE` is unspecified).

Usage Notes

1. If `ODMS_SAMPLING` is unspecified or set as `ODMS_SAMPLING_DISABLE`, the sampling is not performed for AR and the exact AR is obtained.
2. If `ODMS_SAMPLING` is set as `ODMS_SAMPLING_ENABLE` and if `ODMS_SAMPLE_SIZE` is specified as positive integer number then the user-specified sample size (`ODMS_SAMPLE_SIZE`) is utilized. The sampling is performed in the general data preparation stage before the AR algorithm. The AR-specific sample size is not computed. The approximated AR is obtained.
3. If `ODMS_SAMPLING` is set as `ODMS_SAMPLING_ENABLE` and `ODMS_SAMPLE_SIZE` is not specified, the AR-specified sample size is computed and then sampling is performed in the AR algorithm. The approximated AR is obtained.

> **Note:**
>
> If the computed AR-specific sample size is larger than or equal to the total transaction size in the dataset, the sampling is not performed and the exact AR is obtained.

If users do not have a good idea on the choice of sample size for AR, it is suggested to leave `ODMS_SAMPLE_SIZE` unspecified, only specify proper values for sampling parameters and let AR algorithm compute the suitable AR-specific sample size.

> **See Also:**
>
> DBMS_DATA_MINING — Machine Learning Function Settings for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

14.4 Calculating Association Rules

The first step in association analysis is the enumeration of **itemsets**. An itemset is any combination of two or more items in a transaction.

14.4.1 Itemsets

Learn about itemsets.

The maximum number of items in an itemset is user-specified. If the maximum is two, then all the item pairs are counted. If the maximum is greater than two, then all the item pairs, all the item triples, and all the item combinations up to the specified maximum are counted.

The following table shows the itemsets derived from the transactions shown in the following example, assuming that maximum number of items in an itemset is set to 3.

Table 14-1 Itemsets

Transaction	Itemsets
11	(B,D) (B,E) (D,E) (B,D,E)
12	(A,B) (A,C) (A,E) (B,C) (B,E) (C,E) (A,B,C) (A,B,E) (A,C,E) (B,C,E)
13	(B,C) (B,D) (B,E) (C,D) (C,E) (D,E) (B,C,D) (B,C,E) (B,D,E) (C,D,E)

Example 14-1 Sample Transactional Data

```
TRANS_ID   ITEM_ID
---------  -------------------
11         B
11         D
11         E
12         A
12         B
12         C
12         E
13         B
13         C
13         D
13         E
```

14.4.2 Frequent Itemsets

Learn about frequent itemsets and support.

Association rules are calculated from itemsets. If rules are generated from all possible itemsets, there can be a very high number of rules and the rules may not be very meaningful. Also, the model can take a long time to build. Typically it is desirable to only generate rules from itemsets that are well-represented in the data. **Frequent itemsets** are those that occur with a minimum frequency specified by the user.

The minimum frequent itemset **support** is a user-specified percentage that limits the number of itemsets used for association rules. An itemset must appear in at least this percentage of all the transactions if it is to be used as a basis for rules.

The following table shows the itemsets from Table 14-1 that are frequent itemsets with support > 66%.

Table 14-2 Frequent Itemsets

Frequent Itemset	Transactions	Support
(B,C)	2 of 3	67%
(B,D)	2 of 3	67%
(B,E)	3 of 3	100%
(C,E)	2 of 3	67%
(D,E)	2 of 3	67%
(B,C,E)	2 of 3	67%
(B,D,E)	2 of 3	67%

Related Topics

- About Apriori
  Learn how to find associations involving rare events in a large number of items using Apriori.

14.4.3 Example: Calculating Rules from Frequent Itemsets

Example to calculating rules from frequent itemsets.

The following tables show the itemsets and frequent itemsets that were calculated in "Association". The frequent itemsets are the itemsets that occur with a minimum support of 67%; at least 2 of the 3 transactions must include the itemset.

Table 14-3 Itemsets

Transaction	Itemsets
11	(B,D) (B,E) (D,E) (B,D,E)
12	(A,B) (A,C) (A,E) (B,C) (B,E) (C,E) (A,B,C) (A,B,E) (A,C,E) (B,C,E)
13	(B,C) (B,D) (B,E) (C,D) (C,E) (D,E) (B,C,D) (B,C,E) (B,D,E) (C,D,E)

Table 14-4 Frequent Itemsets with Minimum Support 67%

Itemset	Transactions	Support
(B,C)	12 and 13	67%
(B,D)	11 and 13	67%
(B,E)	11, 12, and 13	100%
(C,E)	12 and 13	67%
(D,E)	11 and 13	67%
(B,C,E)	12 and 13	67%
(B,D,E)	11 and 13	67%

A rule expresses a conditional probability. Confidence in a rule is calculated by dividing the probability of the items occurring together by the probability of the occurrence of the antecedent.

For example, if B (antecedent) is present, what is the chance that C (consequent) is also present? What is the confidence for the rule "IF B, THEN C"?

As shown in Table 14-3:

- All 3 transactions include B (3/3 or 100%)
- Only 2 transactions include both B and C (2/3 or 67%)
- Therefore, the confidence of the rule "IF B, THEN C" is 67/100 or 67%.

The following table the rules that can be derived from the frequent itemsets in Table 14-4.

Table 14-5 Frequent Itemsets and Rules

Frequent Itemset	Rules	prob(antecedent and consequent) / prob(antecedent)	Confidence
(B,C)	(If B then C) (If C then B)	67/100 67/67	67% 100%
(B,D)	(If B then D) (If D then B)	67/100 67/67	67% 100%
(B,E)	(If B then E) (If E then B)	100/100 100/100	100% 100%
(C,E)	(If C then E) (If E then C)	67/67 67/100	100% 67%
(D,E)	(If D then E) I(f E then D)	67/67 67/100	100% 67%
(B,C,E)	(If B and C then E) (If B and E then C) (If C and E then B)	67/67 67/100 67/67	100% 67% 100%
(B,D,E)	(If B and D then E) (If B and E then D) (If D and E then B)	67/67 67/100 67/67	100% 67% 100%

If the minimum confidence is 70%, ten rules are generated for these frequent itemsets. If the minimum confidence is 60%, sixteen rules are generated.

Tip:

Increase the minimum confidence if you want to decrease the build time for the model and generate fewer rules.

Related Topics

- About Association
  Association is a Oracle Machine Learning for SQL function that discovers the probability of the co-occurrence of items in a collection.

14.4.4 Aggregates

Aggregates refer to the quantities associated with each item that the user opts for association rules model to aggregate.

There can be more than one aggregate. For example, the user can specify the model to aggregate both profit and quantity.

14.4.5 Example: Calculating Aggregates

This example shows how to calculate aggregates using the customer grocery purchase and profit data.

Calculating Aggregates for Grocery Store Data

Assume a grocery store has the following data:

Table 14-6 Grocery Store Data

Customer	Item A	Item B	Item C	Item D
Customer 1	Buys (Profit $5.00)	Buys (Profit $3.20)	Buys (Profit $12.00)	NA
Customer 2	Buys (Profit $4.00)	NA	Buys (Profit $4.20)	NA
Customer 3	Buys (Profit $3.00)	Buys (Profit $10.00)	Buys (Profit $14.00)	Buys (Profit $8.00)
Customer 4	Buys (Profit $2.00)	NA	NA	Buys (Profit $1.00)

The basket of each customer can be viewed as a transaction. The manager of the store is interested in not only the existence of certain association rules, but also in the aggregated profit if such rules exist.

In this example, one of the association rules can be (A, B)=>C for customer 1 and customer 3. Together with this rule, the store manager may want to know the following:

- The total profit of item A appearing in this rule
- The total profit of item B appearing in this rule
- The total profit for consequent C appearing in this rule
- The total profit of all items appearing in the rule

For this rule, the profit for item A is $5.00 + $3.00 = $8.00, for item B the profit is $3.20 + $10.00 = $13.20, for consequent C, the profit is $12.00 + $14.00 = $26.00, for the antecedent itemset (A, B) is $8.00 + $13.20 = $21.20. For the whole rule, the profit is $21.20 + $26.00 = $47.40.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

14.4.6 Including and Excluding Rules

Explains including rules and excluding rules used in association.

Including rules enables a user to provide a list of items such that at least one item from the list must appear in the rules that are returned. Excluding rules enables a user to provide a list of items such that no item from the list can appear in the rules that are returned.

> **Note:**
>
> Since each association rule includes both antecedent and consequent, a set of including or excluding rules can be specified for antecedent while another set of including or excluding rules can be specified for consequent. Including or excluding rules can also be defined for the association rule.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*
- *Oracle Database PL/SQL Packages and Types Reference*

14.4.7 Performance Impact for Aggregates

Aggregate function requires more memory usage and longer execution time.

For each item, the user may supply several columns to aggregate. It requires more memory to buffer the extra data and more time to compute the aggregate values.

14.5 Evaluating Association Rules

Evaluate association rules by using support and confidence.

Minimum support and confidence are used to influence the build of an association model. Support and confidence are also the primary metrics for evaluating the quality of the rules generated by the model. Additionally, Oracle Machine Learning for SQL supports lift for association rules. These statistical measures can be used to rank the rules and hence the usefulness of the predictions.

14.5.1 Support

The support of a rule indicates how frequently the items in the rule occur together. For example, cereal and milk might appear together in 40% of the transactions. If so, the following rules each have a support of 40%:

```
cereal implies milk
milk implies cereal
```

Support is the ratio of transactions that include all the items in the antecedent and consequent to the number of total transactions.

Support can be expressed in probability notation as follows:

```
support(A implies B) = P(A, B)
```

14.5.2 Minimum Support Count

Minimum support count defines minimum threshold in transactions that each rule must satisfy.

When the number of transactions is unknown, the support percentage threshold parameter can be tricky to set appropriately. For this reason, support can also be expressed as a count of transactions, with the greater of the two thresholds being used to filter out infrequent itemsets. The default is `1` indicating that this criterion is not applied.

Related Topics

- Association Rules
  Identifies the pattern of association within the data.
- *Oracle Machine Learning for SQL User's Guide*
- Frequent Itemsets
  Learn about frequent itemsets and support.

14.5.3 Confidence

The confidence of a rule indicates the probability of both the antecedent and the consequent appearing in the same transaction.

Confidence is the conditional probability of the consequent given the antecedent. For example, cereal appears in 50 transactions; 40 of the 50 might also include milk. The rule confidence is:

```
cereal implies milk with 80% confidence
```

Confidence is the ratio of the rule support to the number of transactions that include the antecedent.

Confidence can be expressed in probability notation as follows.

```
confidence (A implies B) = P (B/A), which is equal to P(A, B) / P(A)
```

Related Topics

- Confidence
- Frequent Itemsets
  Learn about frequent itemsets and support.

14.5.4 Reverse Confidence

The reverse confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs.

Reverse confidence eliminates rules that occur because the consequent is frequent. The default is `0`.

Related Topics

- Confidence

- Example: Calculating Rules from Frequent Itemsets
  Example to calculating rules from frequent itemsets.
- *Oracle Machine Learning for SQL User's Guide*
- *Oracle Database PL/SQL Packages and Types Reference*

14.5.5 Lift

Both support and confidence must be used to determine if a rule is valid. However, there are times when both of these measures may be high, and yet still produce a rule that is not useful. For example:

```
Convenience store customers who buy orange juice also buy milk with
a 75% confidence.
The combination of milk and orange juice has a support of 30%.
```

This at first sounds like an excellent rule, and in most cases, it would be. It has high confidence and high support. However, what if convenience store customers in general buy milk 90% of the time? In that case, orange juice customers are actually *less* likely to buy milk than customers in general.

A third measure is needed to evaluate the quality of the rule. Lift indicates the strength of a rule over the random co-occurrence of the antecedent and the consequent, given their individual support. It provides information about the improvement, the increase in probability of the consequent given the antecedent. Lift is defined as follows.

```
(Rule Support) /(Support(Antecedent) * Support(Consequent))
```

This can also be defined as the confidence of the combination of items divided by the support of the consequent. So in our milk example, assuming that 40% of the customers buy orange juice, the improvement would be:

```
30% / (40% * 90%)
```

which is 0.83 – an improvement of less than 1.

Any rule with an improvement of less than 1 does not indicate a real cross-selling opportunity, no matter how high its support and confidence, because it actually offers less ability to predict a purchase than does random chance.

> **Tip:**
>
> Decrease the maximum rule length if you want to decrease the build time for the model and generate simpler rules.

> **Tip:**
>
> Increase the minimum support if you want to decrease the build time for the model and generate fewer rules.

15
CUR Matrix Decomposition

Learn how to use CUR decomposition based algorithm for attribute importance.

- About CUR Matrix Decomposition
- Singular Vectors
- Statistical Leverage Score
- Column (Attribute) Selection and Row Selection
- CUR Matrix Decomposition Algorithm Configuration

Related Topics

- Feature Selection
  Learn how to perform feature selection and attribute importance.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: CUR Matrix Decomposition
- Automatic Data Preparation
- Model Detail Views for CUR Matrix Decomposition
- OML4SQL Examples

15.1 About CUR Matrix Decomposition

CUR Matrix Decomposition is a low-rank matrix decomposition algorithm that is explicitly expressed in a small number of actual columns and/or actual rows of data matrix.

CUR Matrix Decomposition was developed as an alternative to Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). CUR Matrix Decomposition selects columns and rows that exhibit high **statistical leverage** or large **influence** from the data matrix. By implementing the CUR Matrix Decomposition algorithm, a small number of most important attributes and/or rows can be identified from the original data matrix. Therefore, CUR Matrix Decomposition is an important tool for exploratory data analysis. CUR Matrix Decomposition can be applied to a variety of areas and facilitates regression, classification, and clustering.

Related Topics

- Data Preparation for SVD
  Oracle Machine Learning for SQL implements Singular Value Decomposition (SVD) for numerical data and categorical data.

15.2 Singular Vectors

Singular Value Decomposition (SVD) is the first step in CUR Matrix Decomposition.

SVD returns left and right singular vectors for calculating column and row leverage scores. Perform SVD on the following matrix:

$A \; \varepsilon \; \mathbf{R}^{m \times n}$

The matrix is factorized as follows:

$A = U\Sigma V^T$

where $U = [u^1 \; u^2 \ldots u^m]$ and $V = [v^1 \; v^2 \ldots v^n]$ are orthogonal matrices.

Σ is a diagonal m × n matrix with non-negative real numbers $\sigma 1, \ldots, \sigma_\rho$ on the diagonal, where $\rho = \min \{m, n\}$ and σ_ξ is the ξ^{th} singular value of *A*.

Let u^ξ and v^ξ be the ξ^{th} left and right singular vector of *A*, the j^{th} column of A can thus be approximated by the top *k* singular vectors and corresponding singular values as:

$$A^j \approx \sum_{\xi=1}^{k} \left(\sigma_\xi u^\xi\right) v_j^\xi$$

where v^ξ_j is the j^{th} coordinate of the ξ^{th} right singular vector.

15.3 Statistical Leverage Score

Leverage scores are statistics that determine which column (or rows) are most representative with respect to a rank subspace of a matrix. The statistical leverage scores represent the column (or attribute) and row importance.

The normalized statistical leverage scores for all columns are computed from the top *k* right singular vectors as follows:

$$\pi_j = \frac{1}{k} \sum_{\xi=1}^{k} (v_j^\xi)^2$$

where *k* is called rank parameter and $j = 1, \ldots, n$. Given that π_j>=0 and

$$\sum_{j=1}^{n} \pi_j = 1$$

, these scores form a probability distribution over the *n* columns.

Similarly, the normalized statistical leverage scores for all rows are computed from the top *k* left singular vectors as:

$$\pi'_i = \frac{1}{k}\sum_{\xi=1}^{k}(u_i^{\xi})^2$$

where $i = 1,...,m$.

15.4 Column (Attribute) Selection and Row Selection

The CUR matrix decomposition in OML4SQL is designed for attribute and/or row importance. It returns attributes and rows with high importance that are ranked by their leverage (importance) scores. Column (Attribute) selection and row selection is the final stage in CUR Matrix Decomposition.

Attribute selection: Selects attributes with high leverage scores and reports their names, scores (as importance) and ranks (by importance).

Row selection: Selects rows with high leverage scores and reports their names, scores (as importance) and ranks (by importance).

1. CUR Matrix Decomposition first selects the j^{th} column (or attribute) of *A* with probability $p_j = \min\{1, c\pi_j\}$ for all $j \in \{1,...,n\}$
2. If users enable row selection, select i^{th} row of *A* with probability $p'_i = \min\{1, r\pi'_i\}$ for all $i \in \{1,...,m\}$
3. Report the name (or ID) and leverage score (as importance) for all selected attributes (if row importance is disabled) or for all selected attributes and rows (if row importance is enabled).

c is the approximated (or expected) number of columns that users want to select, and *r* is the approximated (or expected) number of rows that users want to select.

To realize column and row selections, you need to calculate the probability to select each column and row.

Calculate the probability for each column as follows:

$p_j = \min\{1, c\pi_j\}$

Calculate the probability for each row as follows:

$p'_i = \min\{1, c\pi'_i\}$.

A column or row is selected if the probability is greater than some threshold.

15.5 CUR Matrix Decomposition Algorithm Configuration

Configure the CUR Matrix Decomposition algorithm setting to build your model.

Create a model with the algorithm specific settings. Define the algorithm name as `ALGO_CUR_DECOMPOSITION` and mining function as `ATTRIBUTE_IMPORTANCE`.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: CUR Matrix Decomposition for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Row Selection

To use this feature, specify the row importance setting `CURS_ROW_IMPORTANCE` to `CURS_ROW_IMP_ENABLE`.

> **Note:**
>
> The row selection is performed only when users specify that row importance is enabled and the `CASE_ID` column is present.

16
Decision Tree

Oracle Machine Learning for SQL supports Decision Tree as one of the classification algorithms. This chapter provides an overview of the Decision Tree algorithm.

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Decision Tree
- Automatic Data Preparation
- Model Detail Views for Decision Tree
- OML4SQL Examples
- OML4R Decision Tree Example
- OML4R GitHub Examples

16.1 About Decision Tree

Decision tree is a supervised machine learning algorithm used for classifying data. Decision tree has a tree structure built top-down that has a root node, branches, and leaf nodes.

In some applications of Oracle Machine Learning for SQL, the reason for predicting one outcome or another may not be important in evaluating the overall quality of a model. In others, the ability to explain the reason for a decision can be crucial. You can use decision tree rules to validate models in such problems. The Decision Tree algorithm, like Naive Bayes, is based on conditional probabilities. Unlike Naive Bayes, decision trees generate **rules**. A rule is a conditional statement that can be understood by humans and used within a database to identify a set of records.

For example, a Marketing professional requires complete descriptions of customer segments to launch a successful marketing campaign. The Decision Tree algorithm is ideal for this type of application.

Use decision tree rules to validate models. If the rules make sense to a subject matter expert, then this validates the model.

16.1.1 Decision Tree Rules

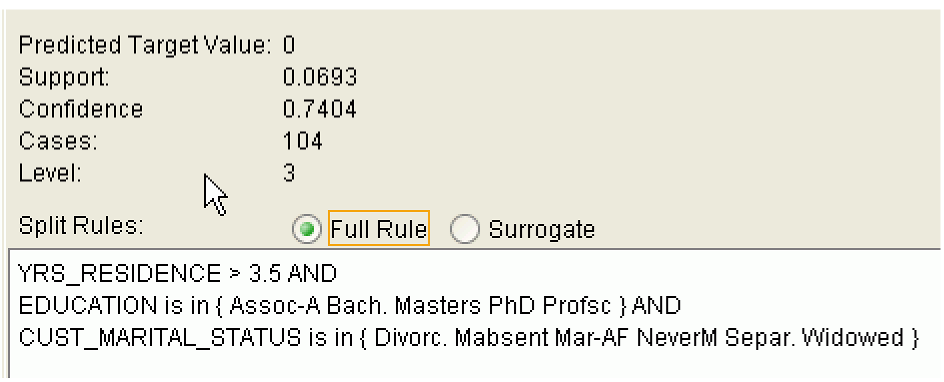
Introduces decision tree rules.

Oracle Machine Learning for SQL supports several algorithms that provide rules. In addition to decision trees, clustering algorithms provide rules that describe the conditions shared by the members of a cluster, and association rules provide rules that describe associations between attributes.

Rules provide **model transparency**, a window on the inner workings of the model. Rules show the basis for the model's predictions. Oracle Machine Learning for SQL supports a high level of model transparency. While some algorithms provide rules, *all* algorithms provide **model details**. You can examine model details to determine how the algorithm handles the attributes internally, including transformations and reverse transformations. Transparency is discussed in the context of data preparation and in the context of model building in *Oracle Machine Learning for SQL User’s Guide*.

The following figure shows a rule generated by a Decision Tree model. This rule comes from a decision tree that predicts the probability that customers increase spending if given a loyalty card. A target value of 0 means not likely to increase spending; 1 means likely to increase spending.

Figure 16-1 Sample Decision Tree Rule



The rule shown in the figure represents the conditional statement:

```
IF
          (current residence > 3.5 and has college degree and is single)
THEN
          predicted target value = 0
```

This rule is a full rule. A surrogate rule is a related attribute that can be used at apply time if the attribute needed for the split is missing.

Related Topics

- Understanding Reverse Transformations
- Model Detail Views for Decision Tree
- About Clustering
  Clustering analysis finds clusters of data objects that are similar to one another.

- About Association
  Association is a Oracle Machine Learning for SQL function that discovers the probability of the co-occurrence of items in a collection.

16.1.1.1 Confidence and Support

Confidence and support are properties of rules. These statistical measures can be used to rank the rules and hence the predictions.

Support: The number of records in the training data set that satisfy the rule.

Confidence: The likelihood of the predicted outcome, given that the rule has been satisfied.

For example, consider a list of 1000 customers (1000 cases). Out of all the customers, 100 satisfy a given rule. Of these 100, 75 are likely to increase spending, and 25 are not likely to increase spending. The **support of the rule** is 100/1000 (10%). The **confidence of the prediction** (likely to increase spending) for the cases that satisfy the rule is 75/100 (75%).

16.1.2 Advantages of Decision Trees

Learn about the advantages of the Decision Tree algorithm.

The Decision Tree algorithm produces accurate and interpretable models with relatively little user intervention. The algorithm can be used for both binary and multiclass classification problems.

The algorithm is fast, both at build time and apply time. The build process for Decision Tree supports parallel execution. (Scoring supports parallel execution irrespective of the algorithm.)

Decision Tree scoring is especially fast. The tree structure, created in the model build, is used for a series of simple tests, (typically 2-7). Each test is based on a single predictor. It is a membership test: either IN or NOT IN a list of values (categorical predictor); or LESS THAN or EQUAL TO some value (numeric predictor).

Related Topics

- *Oracle Database VLDB and Partitioning Guide*

16.1.3 XML for Decision Tree Models

Learn about generating XML representation of Decision Tree models.

You can generate XML representing a Decision Tree model; the generated XML satisfies the definition specified in the Predictive Model Markup Language (PMML) version 2.1 specification.

Related Topics

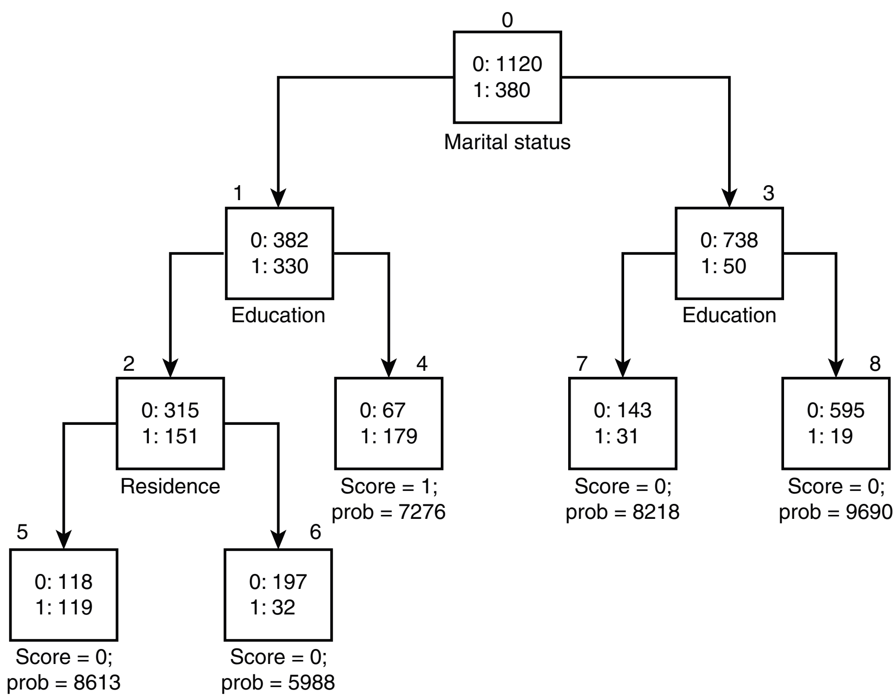
- https://dmg.org

16.2 Growing a Decision Tree

Predict a target value by a sequence of questions to form or grow a decision tree. A sample here shows how to grow a decision tree.

A decision tree predicts a target value by asking a sequence of questions. At a given stage in the sequence, the question that is asked depends upon the answers to the previous questions. The goal is to ask questions that, taken together, uniquely identify specific target values. Graphically, this process forms a tree structure.

Figure 16-2 Sample Decision Tree



The figure is a decision tree with nine nodes (and nine corresponding rules). The target attribute is binary: 1 if the customer increases spending, 0 if the customer does not increase spending. The first split in the tree is based on the `CUST_MARITAL_STATUS` attribute. The root of the tree (node 0) is split into nodes 1 and 3. Married customers are in node 1; single customers are in node 3.

The rule associated with node 1 is:

```
Node 1 recordCount=712,0 Count=382, 1 Count=330
CUST_MARITAL_STATUS isIN  "Married",surrogate:HOUSEHOLD_SIZE isIn "3""4-5"
```

Node 1 has 712 records (cases). In all 712 cases, the `CUST_MARITAL_STATUS` attribute indicates that the customer is married. Of these, 382 have a target of 0 (not likely to increase spending), and 330 have a target of 1 (likely to increase spending).

16.2.1 Splitting

The Decision Tree algorithm offers metrics for splitting the cases (records).

During the training process, the Decision Tree algorithm must repeatedly find the most efficient way to split a set of cases (records) into two child nodes. Oracle Machine Learning for SQL offers two homogeneity metrics, **gini** and **entropy**, for calculating the splits. The default metric is gini.

Homogeneity metrics asses the quality of alternative split conditions and select the one that results in the most homogeneous child nodes. Homogeneity is also called **purity**; it refers to the degree to which the resulting child nodes are made up of cases with the same target value. The objective is to maximize the purity in the child nodes. For example, if the target can be either yes or no (does or does not increase spending), the objective is to produce nodes where most of the cases either increase spending or most of the cases do not increase spending.

16.2.2 Cost Matrix

Learn about a cost matrix for the Decision Tree algorithm.

All classification algorithms, including Decision Tree, support a cost-benefit matrix at apply time. You can use the same cost matrix for building and scoring a decision tree model, or you can specify a different cost/benefit matrix for scoring.

Related Topics

- Costs
- Priors and Class Weights
  Learn about Priors and Class Weights in a classification model to produce a useful result.

16.2.3 Preventing Over-Fitting

Understand over-fitting in trees and what can you do to resolve over-fitting.

In principle, the Decision Tree algorithm can grow each branch of the tree deeply enough to perfectly classify the training examples. While this is sometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. In either of these cases, this simple algorithm can produce trees that over-fit the training examples. Over-fit is a condition where a model is able to accurately predict the data used to create the model, but does poorly on new data presented to it.

To prevent over-fitting, Oracle Machine Learning for SQL supports automatic **pruning** and configurable **limit conditions** that control tree growth. Limit conditions prevent further splits once the conditions have been satisfied. Pruning removes branches that have insignificant predictive power.

16.3 Tuning the Decision Tree Algorithm

Fine tune the Decision Tree algorithm with various parameters.

The Decision Tree algorithm is implemented with reasonable defaults for splitting and termination criteria. However several build settings are available for fine tuning.

You can specify a homogeneity metric for finding the optimal split condition for a tree. The default metric is gini. The entropy metric is also available.

Settings for controlling the growth of the tree are also available. You can specify the maximum depth of the tree, the minimum number of cases required in a child node, the minimum number of cases required in a node in order for a further split to be possible, the minimum number of cases in a child node, and the minimum number of cases required in a node in order for a further split to be possible.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

The training data attributes are binned as part of the algorithm's data preparation. You can alter the number of bins used by the binning step. There is a trade-off between the number of bins used and the time required for the build.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: Decision Tree for a listing and description of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

16.4 Data Preparation for Decision Tree

The Decision Tree algorithm manages its own data preparation internally. It does not require pretreatment of the data.

Decision Tree is not affected by Automatic Data Preparation (ADP).

Related Topics

- Prepare the Data

17 Expectation Maximization

Learn how to use expectation maximization clustering algorithm.

- About Expectation Maximization
- Algorithm Enhancements
- Configuring the Algorithm
- Data Preparation for Expectation Maximization

Related Topics

- Clustering Algorithms
  Learn different clustering algorithms used in Oracle Machine Learning for SQL.
- Expectation Maximization for Anomaly Detection
  An object is identified as an outlier in an EM Anomaly model if its anomaly probability is greater than 0.5.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Expectation Maximization
- Automatic Data Preparation
- Model Detail Views for Expectation Maximization
- OML4SQL Examples
- OML4R Expectation Maximization Example
- OML4R GitHub Examples

17.1 About Expectation Maximization

Expectation maximization (EM) estimation of mixture models is a popular probability density estimation technique that is used in a variety of applications.

Oracle Machine Learning for SQL uses EM to implement a distribution-based clustering algorithm (EM-clustering) and a distribution-based anomaly detection algorithm (EM Anomaly).

17.1.1 Expectation Step and Maximization Step

The two steps to compute the likelihood of the current model and to maximize the likelihood defines the algorithm.

Expectation maximization is an iterative method. It starts with an initial parameter guess. The parameter values are used to compute the likelihood of the current model. This is the Expectation step. The parameter values are then recomputed to maximize the likelihood. This is the Maximization step. The new parameter estimates are used to compute a new expectation and then they are optimized again to maximize the likelihood. This iterative process continues until model convergence.

17.1.2 Probability Density Estimation

You can compute reliable cluster assignment using probability density.

In density estimation, the goal is to construct a density function that captures how a given population is distributed. In probability density estimation, the density estimate is based on observed data that represents a sample of the population. Areas of high data density in the model correspond to the peaks of the underlying distribution.

Density-based clustering is conceptually different from distance-based clustering (for example *k*-Means) where emphasis is placed on minimizing inter-cluster and maximizing the intra-cluster distances. Due to its probabilistic nature, density-based clustering can compute reliable probabilities in cluster assignment. It can also handle missing values automatically.

A distribution-based anomaly detection algorithm identifies an object as an outlier if its probability density is lower than the density of other data records in a data set. The EM Anomaly algorithm can capture the underlying data distribution and thus flag records that do not fit the learned data distribution well.

17.2 Algorithm Enhancements

Expectation Maximization (EM) is enhanced to resolve some challenges in its standard form.

Although EM is well established as a distribution-based algorithm, it presents some challenges in its standard form. The Oracle Machine Learning for SQL implementation includes significant enhancements, such as scalable processing of large volumes of data and automatic parameter initialization. The strategies that OML4SQL uses to address the inherent limitations of EM clustering and EM Anomaly are described further.

> **Note:**
>
> The EM abbreviation is used here to refer to general EM technique for probability density estimation that is common for both EM Clustering and EM Anomaly.

Limitations of Standard Expectation Maximization:

- Scalability: EM has linear scalability with the number of records and attributes. The number of iterations to convergence tends to increase with growing data size (both rows and columns). EM convergence can be slow for complex problems and can place a significant load on computational resources.
- High dimensionality: EM has limited capacity for modeling high dimensional (wide) data. The presence of many attributes slows down model convergence, and the algorithm becomes less able to distinguish between meaningful attributes and noise. The algorithm is thus compromised in its ability to find correlations.

- Number of components: EM typically requires the user to specify the number of components. In most cases, this is not information that the user can know in advance.
- Parameter initialization: The choice of appropriate initial parameter values can have a significant effect on the quality of the model. Initialization strategies that have been used for EM have generally been computationally expensive.
- From components to clusters: In EM Clustering model, components are often treated as clusters. This approach can be misleading since cohesive clusters are often modeled by multiple components. Clusters that have a complex shape need to be modeled by multiple components. To accomplish this, the Oracle Machine Learning for SQL implementation of EM Clustering creates a component hierarchy based on the overlap of the distributions of the individual components. The OML4SQL EM Clustering algorithm employs agglomerative hierarchical clustering. The OML4SQL implementation of EM Custering produces an assignment of the model components to high-level clusters.
- Anomaly Detection: In EM Anomaly detection, an anomaly probability is used to classify whether an object is normal or anomalous. The EM algorithm estimates the probability density of a data record which is mapped to a probability of an anomaly.

17.2.1 Scalability

Expectation Maximization (EM) in Oracle Machine Learning for SQL, uses database parallel processing to achieve excellent scalability.

The OML4SQL implementation of Expectation Maximization uses database parallel processing to achieve excellent scalability. EM computations naturally lend themselves to row parallel processing, and the partial results are easily aggregated. The parallel implementation efficiently distributes the computationally intensive work across secondary processes and then combines the partial results to produce the final solution.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*

17.2.2 High Dimensionality

Process high dimensional data through Expectation Maximization.

The Oracle Machine Learning for SQL implementation of Expectation Maximization (EM) can efficiently process high-dimensional data with thousands of attributes. This is achieved through a two-fold process:

- The data space of single-column (not nested) attributes is analyzed for pair-wise correlations. Only attributes that are significantly correlated with other attributes are included in the EM mixture model. The algorithm can also be configured to restrict the dimensionality to the *M* most correlated attributes.
- High-dimensional (nested) numerical data that measures events of similar type is projected into a set of low-dimensional features that are modeled by EM. Some examples of high-dimensional, numerical data are: text, recommendations, gene expressions, and market basket data.

17.2.3 Number of Components

The number of EM components are automatically determined.

Typical implementations of Expectation Maximization (EM) require the user to specify the number of model components. This is problematic because users do not generally know the correct number of components. Choosing too many or too few components can lead to over-fitting or under-fitting, respectively.

When model search is enabled, the number of EM components is automatically determined. The algorithm uses a held-aside sample to determine the correct number of components, except in the cases of very small data sets when Bayesian Information Criterion (BIC) regularization is used.

17.2.4 Parameter Initialization

Choosing appropriate initial parameter values can have a significant effect on the quality of the solution.

Expectation maximization (EM) is not guaranteed to converge to the global maximum of the likelihood function but may instead converge to a local maximum. Therefore different initial parameter values can lead to different model parameters and different model quality.

In the process of model search, the EM model is grown independently. As new components are added, their parameters are initialized to areas with poor distribution fit.

17.2.5 From Components to Clusters

Expectation Maximization produces assignment of model components to high-level clusters.

Expectation Maximization (EM) model components are often treated as clusters. However, this approach can be misleading. Cohesive clusters are often modeled by multiple components. The shape of the probability density function used in EM effectively predetermines the shape of the identified clusters. For example, Gaussian density functions can identify single peak symmetric clusters. Clusters of more complex shape need to be modeled by multiple components.

Ideally, high density areas of arbitrary shape must be interpreted as single clusters. To accomplish this, the Oracle Machine Learning for SQL implementation of EM builds a component hierarchy that is based on the overlap of the individual components' distributions. OML4SQL EM uses agglomerative hierarchical clustering. Component distribution overlap is measured using the Bhattacharyya distance function. Choosing an appropriate cutoff level in the hierarchy automatically determines the number of high-level clusters.

The OML4SQL implementation of EM produces an assignment of the model components to high-level clusters. Statistics like means, variances, modes, histograms, and rules additionally describe the high-level clusters. The algorithm can be configured to either produce clustering assignments at the component level or at the cluster level.

17.2.6 Expectation Maximization for Anomaly Detection

An object is identified as an outlier in an EM Anomaly model if its anomaly probability is greater than 0.5.

A label of 1 denotes normal, while a label of 0 denotes anomaly. The EM technique models the underlying data distribution of a data set, and the probability density of a data record is translated into an anomaly probability.

The following example displays the code snippet used for anomaly detection using the Expectation Maximization algorithm. Specify the `EMCS_OUTLIER_RATE` setting to capture the desired rate of outliers in the training data set.

```
-- SET OUTLIER RATE IN SETTINGS TABLE - DEFAULT IS 0.05
--

BEGIN DBMS_DATA_MINING.DROP_MODEL('CUSTOMERS360MODEL_AD');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  v_setlst('ALGO_NAME')         := 'ALGO_EXPECTATION_MAXIMIZATION';
  v_setlst('PREP_AUTO')         := 'ON';
  v_setlst('EMCS_OUTLIER_RATE') := '0.1';

  DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'CUSTOMERS360MODEL_AD',
        MINING_FUNCTION     => 'CLASSIFICATION',
        DATA_QUERY          => 'SELECT * FROM CUSTOMERS360_V',
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        SET_LIST            => v_setlst,
        TARGET_COLUMN_NAME  => NULL); -- NULL target indicates anomaly
detection
END;
/
```

To view the complete example, see https://github.com/oracle-samples/oracle-db-examples/blob/main/machine-learning/sql/23c/oml4sql-anomaly-detection-em.sql.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Expectation Maximization

17.3 Configuring the Algorithm

Configure Expectation Maximization (EM).

In Oracle Machine Learning for SQL, EM can effectively model very large data sets (both rows and columns) without requiring the user to supply initialization parameters or specify the number of model components. While the algorithm offers reasonable defaults, it also offers flexibility.

The following list describes some of the configurable aspects of EM:

- Whether or not independent non-nested column attributes are included in the model. For EM Clustering, it is system-determined by default. For EM Anomaly, extreme values in each column attribute can indicate a potential outlier, even when the attribute itself has low dependency on other columns. Therefore, by default the algorithm disables attribute removal in EM Anomaly.
- Whether to use Bernoulli or Gaussian distribution for numerical attributes. By default, the algorithm chooses the most appropriate distribution, and individual attributes may use different distributions. When the distribution is user-specified, it is used for all numerical attributes.
- Whether the convergence criterion is based on a held-aside data set or on Bayesian Information Criterion (BIC). The convergence criterion is system-determined by default.
- The percentage improvement in the value of the log likelihood function that is required to add a new component to the model. The default percentage is 0.001.
- For EM Clustering, whether to define clusters as individual components or groups of components. Clusters are associated to groups of components by default.
- The maximum number of components in the model. If model search is enabled, the algorithm determines the number of components based on improvements in the likelihood function or based on regularization (BIC), up to the specified maximum.
- For EM Clustering, whether the linkage function for the agglomerative clustering step uses the nearest distance within the branch (single linkage), the average distance within the branch (average linkage), or the maximum distance within the branch (complete linkage). By default, the algorithm uses single linkage.
- For EM Anomaly, whether to specify the percentage of the data that is expected to be anomalous. If it is known in advance that the number of "suspicious" cases is a certain percentage of the data, then the outlier rate can be set to that percentage. The algorithm's default value is 0.05.

See Also:

DBMS_DATA_MINING —Algorithm Settings: Expectation Maximization for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

Related Topics

- DBMS_DATA_MINING - Global Settings

17.4 Data Preparation for Expectation Maximization

Learn how to prepare data for Expectation Maximization (EM).

If you use Automatic Data Preparation (ADP), you do not need to specify additional data preparation for Expectation Maximization. ADP normalizes numerical attributes (in non-nested columns) when they are modeled with Gaussian distributions. ADP applies a topN binning transformation to categorical attributes.

Missing value treatment is not needed since Oracle Machine Learning for SQL algorithms handle missing values automatically. The EM algorithm replaces missing values with the mean in single-column numerical attributes that are modeled with Gaussian distributions. In other single-column attributes (categoricals and numericals modeled with Bernoulli distributions), NULLs are not replaced; they are treated as a distinct value with its own frequency count. In nested columns, missing values are treated as zeros.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

18
Explicit Semantic Analysis

Learn how to use Explicit Semantic Analysis (ESA) as an unsupervised algorithm for feature extraction function and as a supervised algorithm for classification.

- About Explicit Semantic Analysis
- Data Preparation for ESA
- Scoring with ESA
- Terminologies in Explicit Semantic Analysis

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- Feature Extraction
  Learn how to perform attribute reduction using feature extraction as an unsupervised function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: Explicit Semantic Analysis
- Automatic Data Preparation
- Model Detail Views for Explicit Semantic Analysis
- OML4SQL Examples
- OML4R Explicit Semantic Analysis Example
- OML4R GitHub Examples

18.1 About Explicit Semantic Analysis

In Oracle Database 12*c* Release 2, Explicit Semantic Analysis (ESA) was introduced as an unsupervised algorithm for feature extraction. Starting from Oracle Database 18c, ESA is enhanced as a supervised algorithm for classification.

As a feature extraction algorithm, ESA does not discover latent features but instead uses explicit features represented in an existing knowledge base. As a feature extraction algorithm, ESA is mainly used for calculating semantic similarity of text documents and for explicit topic modeling. As a classification algorithm, ESA is primarily used for categorizing text documents. Both the feature extraction and classification versions of ESA can be applied to numeric and categorical input data as well.

The input to ESA is a set of attributes vectors. Every attribute vector is associated with a concept. The concept is a feature in the case of feature extraction or a target class in the case of classification. For feature extraction, only one attribute vector may be associated with any feature. For classification, the training set may contain multiple attribute vectors associated with any given target class. These rows related to one target class are aggregated into one by the ESA algorithm.

The output of ESA is a sparse attribute-concept matrix that contains the most important attribute-concept associations. The strength of the association is captured by the weight value of each attribute-concept pair. The attribute-concept matrix is stored as a reverse index that lists the most important concepts for each attribute.

> **Note:**
>
> For feature extraction the ESA algorithm does not project the original feature space and does not reduce its dimensionality. ESA algorithm filters out features with limited or uninformative set of attributes.

The scope of classification tasks that ESA handles is different than the classification algorithms such as Naive Bayes and Support Vector Machine. ESA can perform large scale classification with the number of distinct classes up to hundreds of thousands. The large scale classification requires gigantic training data sets with some classes having significant number of training samples whereas others are sparsely represented in the training data set.

While projecting a document to the ESA topic space produces a high-dimensional sparse vector, it is unsuitable as an input to other machine learning algorithms. Starting from Oracle Database 23c, embeddings are added to address this issue. In natural language processing embeddings refer to a set of language modeling and feature learning techniques in which words, phrases, or documents are mapped to vectors of real numbers. It entails a mathematical transformation from a multi-dimensional space to a continuous vector space with a considerably smaller dimension. Embeddings are usually built on top of an existing knowledge base to gather context data. This method is used to map sparse high-dimensional vectors to dense lower-dimensional vectors while keeping the ESA context available to other machine learning algorithms. The output is a doc2vec (document to vector) mapping, which can be used instead of "bag of words" approach. ESA embeddings allow you to utilize ESA models to generate embeddings for any text or other ESA input. This includes, but is not limited to, embeddings for single words.

To lower the dimensionality of a set of points, a sparse version of the random projection algorithm is utilized. In random projections, the original data is projected into a suitable lower-dimensional space in such a way that the distances between the points are roughly preserved. When compared to other approaches, random projection methods are noted for their power, simplicity, and low error rates. Many natural language tasks apply random projection methods.

```
mining_build_textmining_datadmsh.sqlCREATE_MODEL2


BEGIN DBMS_DATA_MINING.DROP_MODEL('ESA_text_sample_dense');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;

  v_setlst DBMS_DATA_MINING.SETTING_LIST;

BEGIN
  v_setlst('PREP_AUTO')              := 'ON';
  v_setlst('ALGO_NAME')              :=
```

```
'ALGO_EXPLICIT_SEMANTIC_ANALYS';
  v_setlst('ODMS_TEXT_POLICY_NAME')    := 'DMDEMO_ESA_POLICY';
  v_setlst('ESAS_MIN_ITEMS')           := '5';
  v_setlst('ODMS_TEXT_MIN_DOCUMENTS') := '2';
  v_setlst('ESAS_EMBEDDINGS')          := 'ESAS_EMBEDDINGS_ENABLE';
  v_setlst('ESAS_EMBEDDING_SIZE')      := '1024';

  dbms_data_mining_transform.SET_TRANSFORM(
    xformlist, 'comments', null, 'comments', 'comments',
      'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)(TOKEN_TYPE:STEM)');

  DBMS_DATA_MINING.CREATE_MODEL2(
    model_name          => 'ESA_text_sample_dense',
    mining_function     => 'FEATURE_EXTRACTION',
    data_query          => 'SELECT * FROM mining_build_text',
    case_id_column_name => 'cust_id',
    set_list            => v_setlst,
    xform_list          => xformlist);
END;
/
```

To view the complete example, see https://github.com/oracle-samples/oracle-db-examples/blob/main/machine-learning/sql/23c/oml4sql-feature-extraction-text-mining-esa.sql.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Explicit Semantic Analysis

18.1.1 Scoring with ESA

A typical feature extraction application of Explicit Semantic Analysis (ESA) is to identify the most relevant features of a given input and score their relevance. Scoring an ESA model produces data projections in the concept feature space.

If an ESA model is built from an arbitrary collection of documents, then each one is treated as a feature. You can then identify the most relevant documents in the collection. The feature extraction functions are: `FEATURE_DETAILS`, `FEATURE_ID`, `FEATURE_SET`, `FEATURE_VALUE`, and `FEATURE_COMPARE`. The same functions are utilized in the implementation of ESA embeddings, but the space of the features is different. The names of features for ESA embeddings are successive integers starting with 1. The output of `FEATURE_ID` is numeric. Feature IDs in the output of `FEATURE_SET` and `FEATURE_DETAILS` are also numeric.

A typical classification application of ESA is to predict classes of a given document and estimate the probabilities of the predictions. As a classification algorithm, ESA implements the following scoring functions: `PREDICTION`, `PREDICTION_PROBABILITY`, `PREDICTION_SET`, `PREDICTION_DETAILS`, `PREDICTION_COST`.

Related Topics

- *Oracle Machine Learning for SQL User’s Guide*
- *Oracle Database SQL Language Reference*

18.1.2 Scoring Large ESA Models

Building an Explicit Semantic Analysis (ESA) model on a large collection of text documents can result in a model with many features or titles.

The model information for scoring is loaded into System Global Area (SGA) as a shared (shared pool size) library cache object. Different SQL predictive queries can reference this object. When the model size is large, it is necessary to set the SGA parameter in the database to a sufficient size that accommodates large objects. If the SGA is too small, the model may need to be re-loaded every time it is referenced which is likely to lead to performance degradation.

18.2 ESA for Text Analysis

Learn how Explicit Semantic Analysis (ESA) can be used for machine learning operations on text.

Explicit knowledge often exists in text form. Multiple knowledge bases are available as collections of text documents. These knowledge bases can be generic, for example, Wikipedia, or domain-specific. Data preparation transforms the text into vectors that capture attribute-concept associations. ESA is able to quantify semantic relatedness of documents even if they do not have any words in common. The function `FEATURE_COMPARE` can be used to compute semantic relatedness.

Related Topics

- *Oracle Database SQL Language Reference*

18.3 Data Preparation for ESA

Automatic Data Preparation normalizes input vectors to a unit length for Explicit Semantic Analysis (ESA).

When there are missing values in columns with simple data types (not nested), ESA replaces missing categorical values with the mode and missing numerical values with the mean. When there are missing values in nested columns, ESA interprets them as sparse. The algorithm replaces sparse numeric data with zeros and sparse categorical data with zero vectors. The Oracle Machine Learning for SQL data preparation transforms the input text into a vector of real numbers. These numbers represent the importance of the respective words in the text.

See Also:

DBMS_DATA_MINING —Algorithm Settings: Explicit Semantic Analysis for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

18.4 Terminologies in Explicit Semantic Analysis

Discusses the terms associated with Explicit Semantic Analysis (ESA).

Multi-target Classification

The training items in these large scale classifications belong to several classes. The goal of classification in such case is to detect possible multiple target classes for one item. This kind of classification is called multi-target classification. The target column for ESA-based classification is extended. Collections are allowed as target column values. The collection type for the target in ESA-based classification is `ORA_MINING_VARCHAR2_NT`.

Large-scale classification

Large-scale classification applies to ontologies that contain gigantic numbers of categories, usually ranging in tens or hundreds of thousands. This large-scale classification also requires gigantic training datasets which are usually unbalanced, that is, some classes may have significant number of training samples whereas others may be sparsely represented in the training dataset. Large-scale classification normally results in multiple target class assignments for a given test case.

Topic modeling

Topic modelling refers to derivation of the most important topics of a document. Topic modeling can be explicit or latent. Explicit topic modeling results in the selection of the most relevant topics from a pre-defined set, for a given document. Explicit topics have names and can be verbalized. Latent topic modeling identifies a set of latent topics characteristic for a collection of documents. A subset of these latent topics is associated with every document under examination. Latent topics do not have verbal descriptions or meaningful interpretation.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

19

Exponential Smoothing

Learn about the Exponential Smoothing algorithm.

- About Exponential Smoothing
- Data Preparation for Exponential Smoothing Models

Related Topics

- Time Series
  Learn about time series as an Oracle Machine Learning for SQL regression function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing
- Automatic Data Preparation
- Model Detail Views for Exponential Smoothing
- OML4SQL Examples
- OML4R GitHub Examples

19.1 About Exponential Smoothing

Exponential smoothing is a forecasting method for time series data. It is a moving average method where exponentially decreasing weights are assigned to past observations.

Exponential smoothing methods have been widely used in forecasting for over half a century. A forecast is a prediction based on historical data and patterns. preIt has applications at the strategic, tactical, and operation level. For example, at a strategic level, forecasting is used for projecting return on investment, growth and the effect of innovations. At a tactical level, forecasting is used for projecting costs, inventory requirements, and customer satisfaction. At an operational level, forecasting is used for setting targets and predicting quality and conformance with standards.

In its simplest form, exponential smoothing is a moving average method with a single parameter which models an exponentially decreasing effect of past levels on future values. With a variety of extensions, exponential smoothing covers a broader class of models than other well-known approaches, such as the Box-Jenkins auto-regressive integrated moving average (ARIMA) approach. Oracle Machine Learning for SQL implements exponential smoothing using a state of the art state space method that incorporates a single source of error (SSOE) assumption which provides theoretical and performance advantages.

Exponential smoothing is extended to the following:

- A matrix of models that mix and match error type (additive or multiplicative), trend (additive, multiplicative, or none), and seasonality (additive, multiplicative, or none)
- Models with damped trends.
- Models that directly handle irregular time series and time series with missing values.
- Multiple time series models

See Also:

Ord, J.K., et al, *Time Series Forecasting: The Case for the Single Source of Error State Space Approach, Working Paper*, Department of Econometrics and Business Statistics, Monash University, VIC 3800, Australia, April 2, 2005.

19.1.1 Exponential Smoothing Models

Exponential Smoothing models are a broad class of forecasting models that are intuitive, flexible, and extensible.

Members of this class include simple, single parameter models that predict the future as a linear combination of a previous level and a current shock. Extensions can include parameters for linear or non-linear trend, trend damping, simple or complex seasonality, related series, various forms of non-linearity in the forecasting equations, and handling of irregular time series.

Exponential smoothing assumes that a series extends infinitely into the past, but that influence of past on future, decays smoothly and exponentially fast. The smooth rate of decay is expressed by one or more smoothing constants. The **smoothing constants** are parameters that the model estimates. The assumption is made practical for modeling real world data by using an equivalent recursive formulation that is only expressed in terms of an estimate of the current level based on prior history and a shock to that estimate dependent on current conditions only.The procedure requires an estimate for the time period just prior to the first observation, that encapsulates all prior history. This initial observation is an additional model parameter whose value is estimated by the modeling procedure.

Components of ESM such as trend and seasonality extensions, can have an additive or multiplicative form. The simpler additive models assume that shock, trend, and seasonality are linear effects within the recursive formulation.

19.1.2 Simple Exponential Smoothing

Simple exponential smoothing assumes the data fluctuates around a stationary mean, with no trend or seasonal pattern.

In a simple Exponential Smoothing model, each forecast (smoothed value) is computed as the weighted average of the previous observations, where the weights decrease exponentially depending on the value of smoothing constant α. Values of the smoothing constant, α, near one, put almost all weight on the most recent observations. Values of α near zero allows the distant past observations to have a large influence.

19.1.3 Models with Trend but No Seasonality

The preferred form of additive (linear) trend is sometimes called Holt's method or double exponential smoothing.

Models with trend add a smoothing parameter γ and optionally a damping parameter φ. The damping parameter smoothly dampens the influence of past linear trend on future estimates of level, often improving accuracy.

19.1.4 Models with Seasonality but No Trend

When the time series average does not change over time (stationary), but is subject to seasonal fluctuations, the appropriate model has seasonal parameters but no trend.

Seasonal fluctuations are assumed to balance out over periods of length *m*, where *m* is the number of seasons, For example, *m*=4 might be used when the input data are aggregated quarterly. For models with additive errors, the seasonal parameters must sum to zero. For models with multiplicative errors, the product of seasonal parameters must be one.

19.1.5 Models with Trend and Seasonality

Holt and Winters introduced both trend and seasonality in an Exponential Smoothing model.

The original model, also known as Holt-Winters or triple exponential smoothing, considered an additive trend and multiplicative seasonality. Extensions include models with various combinations of additive and multiplicative trend, seasonality and error, with and without trend damping.

19.1.6 Prediction Intervals

To compute prediction intervals, an Exponential Smoothing (ESM) model is divided into three classes.

The simplest class is the class of linear models, which include, among others, simple ESM, Holt's method, and additive Holt-Winters. Class 2 models (multiplicative error, additive components) make an approximate correction for violations of the Normality assumption. Class 3 modes use a simple simulation approach to calculate prediction intervals.

19.2 Data Preparation for Exponential Smoothing Models

Learn about preparing the data for an Exponential Smoothing (ESM) model.

To build an ESM model, you must supply the following :

- Input data
- An aggregation level and method, if the case id is a date type
- Partitioning column, if the data are partitioned

In addition, for a greater control over the build process, the user may optionally specify model build parameters, all of which have defaults:

- Model
- Error type
- Optimization criterion
- Forecast Window
- Confidence level for forecast bounds
- Missing value handling
- Whether the input series is evenly spaced

Related Topics

- *Oracle Machine Learning for SQL User’s Guide*

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: Exponential Smoothing Models for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

19.2.1 Input Data

Time series analysis requires ordered input data. Hence, each data row must consist of an [index, value] pair, where the index specifies the ordering.

When you create an Exponential Smoothing (ESM) model using the `CREATE_MODEL` or the `CREATE_MODEL2` procedure, the `CASE_ID_COLUMN_NAME` and the `TARGET_COLUMN_NAME` parameters are used to specify the columns used to compute the input indices and the observed time series values, respectively. The time column bears Oracle number, or Oracle date, timestamp, timestamp with time zone, or timestamp with local time zone. When the case id column is of type Oracle `NUMBER`, the model considers the input time series to be equally spaced. Only the ordinal position matters, with a lower number indicating a later time. In particular, the input time series is sorted based on the value of `case_id` (time label). The case_id column cannot contain missing values. To indicate a gap, the value column can contain missing values as `NULL`. The magnitude of the difference between adjacent time labels is irrelevant and is not used to calculate the spacing or gap size. Integer numbers passed as `CASE_ID` are assumed to be non-negative.

ESM also supports partitioned models and in such cases, the input table contains an extra column specifying the partition. All [index, value] pairs with the same partition ID form one complete time series. The Exponential Smoothing algorithm constructs models for each partition independently, although all models use the same model settings.

Data properties may result in a warning notice, or settings may be disregarded. If the user sets a model with a multiplicative trend, multiplicative seasonality, or both, and the data contains values Y_t<= 0, the model type is set to default. If the series contains fewer values than the number of seasons given by the user, then the seasonality specifications are ignored and a warning is issued.

If the user has selected a list of predictor series using the parameter `EXSM_SERIES_LIST`, the input data can also include up to twenty additional time series columns.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings:Exponential Smoothing

19.2.2 Accumulation

For the Exponential Smoothing algorithm, the accumulation procedure is applied when the column is a date type (`date`, `datetime`, `timestamp`, `timestamp with timezone`, or `timestamp with local timezone`).

The case id can be a `NUMBER` column whose sort index represents the position of the value in the time series sequence of values. The case id column can also be a date type. A date type is accumulated in accordance with a user specified accumulation window. Regardless of type, the case id is used to transform the column into an equally spaced time series. No accumulation is applied for a case id of type `NUMBER`. As an example, consider a time series about promotion events. The time column contains the date of each event, and the dates can be unequally spaced. The user must specify the spacing interval, which is the spacing of the accumulated or transformed equally spaced time series. In the example, if the user specifies the interval to be month, then an equally spaced time series with profit for each calendar month is generated from the original time series. Setting `EXSM_INTERVAL` is used to specify the spacing interval. The user must also specify a value for `EXSM_ACCUMULATE`, for example, `EXSM_ACCU_MAX`, in which case the equally spaced monthly series would contain the maximum profit over all events that month as the observed time series value.

19.2.3 Missing Value

Input time series can contain missing values. A `NULL` entry in the target column indicates a missing value. When the time column is of the type datetime, the accumulation procedure can also introduce missing values. The setting `EXSM_SETMISSING` can be used to specify how to handle missing values. The special value `EXSM_MISS_AUTO` indicates that, if the series contains missing values it is to be treated as an irregular time series.

> **Note:**
>
> Missing value handling setting must be compatible with model setting, otherwise an error is thrown.

19.2.4 Prediction

An Exponential Smoothing (ESM) model can be applied to make predictions by specifying the prediction window.

Setting `EXSM_PREDICTION_STEP` can be used to specify the prediction window. The prediction window is expressed in terms of number of intervals (setting `EXSM_INTERVAL`), when the time column is of the type datetime. If the time column is a number then the prediction window is the number of steps to forecast. Regardless of whether the time series is regular or irregular, `EXSM_PREDICTION_STEP` specifies the prediction window.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

19.2.5 Parallellism by Partition

Oracle Machine Learning for SQL supports parallellism by partition.

For example, a user can choose `PRODUCT_ID` as one partition column and can generate forecasts for different products in a model build. Although a distinct smoothing model is built for each partition, all partitions share the same model settings. For example, if setting `EXSM_MODEL` is set to `EXSM_SIMPLE`, all partition models will be simple Exponential Smoothing models. Time series from different partitions can be distributed to different processes and processed in parallel. The model for each time series is built serially.

19.2.6 Initial Value Optimization

With long seasonal cycles, users can choose not to optimize the ESM model initial values beyond an initial estimate.

This is in contrast to standard ESM optimization, in which the initial values are adjusted during the optimization process to minimize error. Optimizing only the level, trend, and seasonality parameters rather than the initial values can result in significant performance improvements and faster optimization convergence. When domain knowledge indicates that long seasonal variation is a significant contributor to an accurate forecast, this approach is appropriate. Despite the performance benefits, Oracle does not recommend disabling the optimization of the initial values for typical short seasonal cycles because it may result in model overfitting and less reliable confidence bounds.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings:Exponential Smoothing

19.3 Multiple Time Series Models

Multiple time series is a convenience operation for constructing input to a time series regression. Multiple time series builds multiple time series models with a common time interval for use as input to a time series regression. One of the time series models is identified as the target time series of interest.

All of the time series output is produced for the target. The other time series are assumed to be correlated with the target.

This operation produces backcasts and forecasts on each time series and computes upper and lower confidence bounds for the identified target series. This operation can be used to forecast a wide variety of events, such as rainfall, sales, and customer satisfaction.

In the example of weather forecasting, the temperature and humidity attributes can be considered as the dependent or correlated time series and rainfall can be identified as the target time series.

Related Topics

- Model Detail Views for Exponential Smoothing

19.3.1 Backcasts in Time Series

In the rainfall, temperature, and humidity multiple time series example,backcast are the estimate produced by the model for historical data.

For example, if rainfall is dependent on humidity, then it is useful to have a value of humidity for the period of interest. For periods that have already occurred and are being used to construct the model, such as last week, it is necessary to have the humidity from last week and not from last month.

19.3.2 How to Build Multiple Time Series Models

Oracle's exponential smoothing is enhanced to handle the building of multiple time series models with a single call to the model build method, in addition to single time series forecasting. Multiple time series is built by specifying a series list `EXSM_SERIES_LIST`. The rest of the parameters are the same as in ESM model.

In the weather forecast example, you can have a build data set and a score data set. The build data set contains the identified target series (rain), the dependent series - temperature and humidity. The `DM$VP` model detail view is used to display a forecast for the identified target series (rain), along with dependent series: temperate, and humidity. The `DM$VR` model detail view is used to display backcasts for target series (rain), humidity, and temperature. The backcasts and forecasts of the time series model can be fed into a regression technique like generalized linear model, neural network, or XGBoost for time series regression.

The sample code in the example uses Stock market data that you can download from https://github.com/oracle-samples/oracle-db-examples/blob/main/machine-learning/sql/23c/oml4sql-time-series-regression-dataset.sql and run it.
In the following example, the target attribute DAX is a Stock market index that is being forecast. The dependent attributes that are also popular stock market indexes - SMI, CAC, FTSE are passed as multiple series attributes. Exponential Smoothing settings are used to build a multiple time series model by specifying a series list (`EXSM_SERIES_LIST`) with multiple attributes.

1. Build a multiple time series model.

```
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME')            := 'ALGO_EXPONENTIAL_SMOOTHING';
    v_setlst('EXSM_INTERVAL')        := 'EXSM_INTERVAL_DAY';
    v_setlst('EXSM_MODEL')           := 'EXSM_ADDWINTERS_DAMPED';
    v_setlst('EXSM_SEASONALITY')     := '7';
    v_setlst('EXSM_PREDICTION_STEP') := '1';
    v_setlst('EXSM_SERIES_LIST')     := 'SMI,CAC,FTSE';

    dbms_data_mining.create_model2(
                MODEL_NAME          => 'MSDEMO_MODEL',
                MINING_FUNCTION     => 'TIME_SERIES',
                DATA_QUERY          => 'SELECT * FROM EUSTOCK',
                CASE_ID_COLUMN_NAME => 'DATES',
                TARGET_COLUMN_NAME  => 'DAX',
```

```
                    SET_LIST                => v_setlst);
END;
/
```

2. Use the `DM$VPMSDEMO_MODEL` view to display the forecast.

```
SELECTCASE_ID, VALUE, PREDICTION, UPPER, LOWERFROMDM$VPMSDEMO_MODEL;
```

3. Use the `DM$VRMSDEMO_MODEL` view to display the backcasts.

```
SELECT * FROM DM$VRMSDEMO_MODEL
FETCH FIRST 10 ROWS ONLY;
```

The output of the this model is used in time series regression.

19.4 Time Series Regression

Time series regression is possible with the multi-series build. Time series regression expands the features that can be included in a time series model and possibly improves forecast accuracy. Some of the additional features can be other time series that are thought to be related or dependent to the "target" series.

Temperature and humidity are both dependent time series with rainfall, so by looking at historical data for these two attributes, we can make predictions about future rainfall. When the temperature is high and the humidity is high, there is a greater chance of rainfall.

A time series regression model will take into account the relationship between temperature and humidity, as well as other factors (for example, the location and elevation of the forecast location). The model then produces a prediction for the amount of rainfall (the target series), along with upper and lower bounds. For example, if the model predicts that there is a 90% chance of rain, and the upper bound for the amount of rainfall is 1 inch, then you might want to make sure that you have enough rain gear on hand.

Backcasts can be used to possibly improve the accuracy of forecasts for future time periods. The challenge with using regression to forecast is that the predictors' future values must be given. If, for example, temperature and humidity are the predictors, you need to know their future values on the same time scale as the rainfall series to make a forecast.

Related Topics

- Model Detail Views for Exponential Smoothing

> **See Also:**
>
> Hyndman, R.J. and Athanasopoulos, G., *Forecasting: Principles and Practice*, *3rd edition*, Department of Econometrics and Business Statistics, Monash University, VIC 3800, Australia, May 2021, Chapter 7

19.4.1 How to Build Time Series Regression Models

Oracle exponential smoothing solves the problem of knowing future values on the same time scale as the target series by forecasting the predictor time series using exponential smoothing.

To build a regression model that predicts a future period, the correlated series must have a value in that future period. Hence, all correlated series must be forecast. Backcasts are included for the correlated series as smoothed versions of the correlated series values that can be used as input to the regression model. Backcasts are also available for the target series, as these are part of the standard output of an Oracle machine learning time series model. Target series backcasts can also be included in the regression model.

You can also create build and score datasets. The build data set contains the target series (forecast series), for example, rain; the backcasted target series, for example, backcasted rain; and the backcasted dependent series, for example, backcasted temperature and humidity. The backcasts and forecasts of the time series models can both be used as input to the regression model. The series all use the same time periods, so that the values of the target and the predictors co-occur.

The score data set follows the same schema as the build data set but provides forecasts as required for future values. The score data set can be supplied to the apply procedure of the regression model. Backcasts can be smoother and more structurally consistent with forecasts. The incremental improvement of the regression model over the baseline model can be seen in the backcast of the target series.

Because of the database's versatility, different time series regression variations are possible. A user can add factors such as holidays and environmental changes to the build and score data sets that account for categorical variables. In multiple time series regression, flag variables can be used to account for events or conditions that may have a significant impact on the dependent variable. For example, you might use a flag variable to indicate whether a particular day is a public holiday, or whether a particular month is a winter month. The inclusion of such factors in the model can improve the accuracy of the forecast by accounting for the impact of categorical variables on the dependent variable.

Holidays can be expressed as a binary value column (0s and 1s). For example, a `national_holiday` column can be made that has a value of 1 for national holidays and a value of 0 at other times. In a demand forecast, a perceived change in the environment, like the introduction of a competitor's product, can also be shown as a binary value column, with 0 for times before the introduction and 1 for times after.

Furthermore, as a special case, if a user happens to know the future values of the dependent series, a user could replace the backcasts with the original values in the regression build procedure by creating a data set that joins to the original build table. This user-created data set replaces the build data set.

In the following example, a training, actual, and a test data sets are created using the stock market data. A special case of actual values are provided in the prediction data set to compare the accuracy of ESM and regression. The variable `prod` is a flag variable that accounts for categorical values. It indicates a change in the environment such as an introduction of a new product. The `DM$VR<model_name>` model detail view provides details of the time series regression build schema or the forecast of the target column.

1. Create a build/training data set.

```
BEGIN DROP TABLE tmesm_ms_train;
EXCEPTION WHEN OTHERS THEN NULL; END;
/
CREATE TABLE tmesm_ms_train as
SELECT CASE_ID, DAX, DM$DAX, DM$SMI, DM$CAC, DM$FTSE,
       CASE WHEN case_id < to_date('1998-02-03','YYYY-MM-DD')
       THEN 0 ELSE 1 END AS prod
FROM DM$VRMSDEMO_Model order by 1;
```

2. Create an actual data set (this is the special case scenario where the future values of dependent series is known).

```
BEGIN EXECUTE IMMEDIATE 'DROP TABLE tmesm_ms_actual';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE tmesm_ms_actual (case_id DATE, DAX binary_double);
INSERT INTO tmesm_ms_actual VALUES(DATE '1998-02-04', 4633.008);
commit;
```

3. Query the table to see the output:

```
select * from tmesm_ms_actual;
```

   `CASE_ID` and `DAX` value *4633.00* is displayed.

4. Create a test data set.

```
BEGIN EXECUTE IMMEDIATE 'DROP TABLE tmesm_ms_test';
EXCEPTION WHEN OTHERS THEN NULL; END;
CREATE TABLE tmesm_ms_test as
SELECT a.case_id, b.DAX, DM$DAX, DM$SMI, DM$CAC, DM$FTSE, 1 prod
FROM   DM$VTMSDEMO_model a, tmesm_ms_actual b
WHERE  a.case_id=b.case_id;
```

   The output displays that the procedure completed successfully and a `tmesm_ms_test` table is created.

5. Create a GLM time series regression model using the training data set.

```
SET echo OFF;
BEGIN DBMS_DATA_MINING.DROP_MODEL('MS_GLM_MODEL');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
SET echo ON;
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME')          :=
'ALGO_GENERALIZED_LINEAR_MODEL';

    DBMS_DATA_MINING.CREATE_MODEL2(
```

```
                  MODEL_NAME          => 'MS_GLM_MODEL',
                  MINING_FUNCTION     => dbms_data_mining.regression,
                   DATA_QUERY         => 'SELECT * FROM tmesm_ms_train',
                  CASE_ID_COLUMN_NAME => 'CASE_ID',
                  TARGET_COLUMN_NAME  => 'DAX',
                  SET_LIST            => v_setlst);
END;
/
```

6. Analyze your model by viewing model detail views.

```
SELECT VIEW_NAME, VIEW_TYPE
FROM   USER_MINING_MODEL_VIEWS
WHERE  MODEL_NAME='MSDEMO_MODEL'
ORDER BY VIEW_NAME;
```

7. View the backcasts.

```
SELECT *
FROM   DM$VRMSDEMO_MODEL
ORDER BY CASE_ID
FETCH FIRST 10 ROWS ONLY;
```

8. View the forecast.

```
SELECT *
FROM DM$VTMSDEMO_MODEL
ORDER BY CASE_ID
FETCH FIRST 10 ROWS ONLY;
```

Further, you may compare the baseline (ESM) forecast with that of the regression forecast.

You can view the complete example by accessing `oml4sql-time-series-regression.sql` from https://github.com/oracle-samples/oracle-db-examples/tree/main/machine-learning/sql/23c.

Related Topics

- Model Detail Views for Exponential Smoothing

20

Generalized Linear Model

Learn how to use Generalized Linear Model (GLM) statistical technique for linear modeling.

Oracle Machine Learning for SQL supports GLM for regression and binary classification.

- About Generalized Linear Model
- GLM in Oracle Machine Learning for SQL
- Scalable Feature Selection
- Tuning and Diagnostics for GLM
- GLM Solvers
- Data Preparation for GLM
- Linear Regression
- Logistic Regression

Related Topics

- Regression
  Learn how to predict a continuous numerical target through regression - the supervised machine learning technique.
- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- Feature Selection
  Learn how to perform feature selection and attribute importance.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Generalized Linear Models
- Automatic Data Preparation
- Model Detail Views for Generalized Linear Model
- OML4SQL Examples
- OML4R Generalized Linear Model Example
- OML4R GitHub Examples

20.1 About Generalized Linear Model

The Generalized Linear Model (GLM) includes and extends the class of linear models which address and accommodate some restrictive assumptions of the linear models.

Linear models make a set of restrictive assumptions, most importantly, that the target (dependent variable *y*) is normally distributed conditioned on the value of predictors with a constant variance regardless of the predicted response value. The advantage of linear models and their restrictions include computational simplicity, an interpretable model form, and the ability to compute certain diagnostic information about the quality of the fit.

GLM relaxes these restrictions, which are often violated in practice. For example, binary (yes/no or 0/1) responses do not have same variance across classes. Furthermore, the sum of terms in a linear model typically can have very large ranges encompassing very negative and very positive values. For the binary response example, we would like the response to be a probability in the range [0,1].

GLM accommodates responses that violate the linear model assumptions through two mechanisms: a link function and a variance function. The link function transforms the target range to potentially -infinity to +infinity so that the simple form of linear models can be maintained. The variance function expresses the variance as a function of the predicted response, thereby accommodating responses with non-constant variances (such as the binary responses).

Oracle Machine Learning for SQL includes two of the most popular members of the GLM family of models with their most popular link and variance functions:

- **Linear regression** with the identity link and variance function equal to the constant 1 (constant variance over the range of response values).
- **Logistic regression**

In other words, the methods of linear regression assume that the target value ranges from minus infinity to infinity and that the target variance is constant over the range. The logistic regression target is either 0 or 1. A logistic regression model estimate is a probability. The job of the link function in logistic regression is to transform the target value into the required range, minus infinity to infinity.

GLM Function	Default Link Function	Other Supported Link Functions
Linear regression (gaussian)	identity	none
Logistic regression (binomial)	logit	probit, cloglog, cauchit, and binomial variance

Related Topics

- Linear Regression
- Linear Regression
- Logistic Regression

20.2 GLM in Oracle Machine Learning for SQL

Learn how Oracle Machine Learning for SQL implements the Generalized Linear Model (GLM) algorithm.

GLM is a parametric modeling technique. Parametric models make assumptions about the distribution of the data. When the assumptions are met, parametric models can be more efficient than non-parametric models.

The challenge in developing models of this type involves assessing the extent to which the assumptions are met. For this reason, quality diagnostics are key to developing quality parametric models.

20.2.1 Interpretability and Transparency

You can interpret and understand key characteristics of Generalized Linear Model (GLM) model through model details and global details.

You can interpret Oracle Machine Learnings' GLM with ease. Each model build generates many statistics and diagnostics. Transparency is also a key feature: model details describe key characteristics of the coefficients, and global details provide high-level statistics.

Related Topics

- Tuning and Diagnostics for GLM

20.2.2 Wide Data

Generalized Linear Model(GLM) in Oracle Machine Learning for SQL is uniquely suited for handling wide data. The algorithm can build and score quality models that use a virtually limitless number of predictors (attributes). The only constraints are those imposed by system resources.

20.2.3 Confidence Bounds

Predict confidence bounds through the Generalized Linear Model (GLM) algorithm.

GLM have the ability to predict confidence bounds. In addition to predicting a best estimate and a probability (classification only) for each row, GLM identifies an interval wherein the prediction (regression) or probability (classification) lies. The width of the interval depends upon the precision of the model and a user-specified confidence level.

The confidence level is a measure of how sure the model is that the true value lies within a confidence interval computed by the model. A popular choice for confidence level is 95%. For example, a model might predict that an employee's income is $125K, and that you can be 95% sure that it lies between $90K and $160K. Oracle Machine Learning for SQL supports 95% confidence by default, but that value can be configured.

> **Note:**
>
> Confidence bounds are returned with the coefficient statistics. You can also use the `PREDICTION_BOUNDS` SQL function to obtain the confidence bounds of a model prediction.

Related Topics

- *Oracle Database SQL Language Reference*

20.2.4 Ridge Regression

Understand the use of ridge regression for singularity (exact multicollinearity) in data.

The best regression models are those in which the predictors correlate highly with the target, but there is very little correlation between the predictors themselves. **Multicollinearity** is the term used to describe multivariate regression with correlated predictors.

Ridge regression is a technique that compensates for multicollinearity. Oracle Machine Learning for SQL supports ridge regression for both regression and classification machine learning techniques. The algorithm automatically uses ridge if it detects singularity (exact multicollinearity) in the data.

Information about singularity is returned in the global model details.

Related Topics

- Global Model Statistics for Linear Regression
- Global Model Statistics for Logistic Regression

20.2.4.1 Configuring Ridge Regression

Configure ridge regression through build settings.

You can choose to explicitly enable ridge regression by specifying a build setting for the model. If you explicitly enable ridge, you can use the system-generated ridge parameter or you can supply your own. If ridge is used automatically, the ridge parameter is also calculated automatically.

The configuration choices are summarized as follows:

- Whether or not to override the automatic choice made by the algorithm regarding ridge regression
- The value of the ridge parameter, used only if you specifically enable ridge regression.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

20.2.4.2 Ridge and Confidence Bounds

Models built with ridge regression do not support confidence bounds.

Related Topics

- Confidence Bounds
  Predict confidence bounds through the Generalized Linear Model (GLM) algorithm.

20.2.4.3 Ridge and Data Preparation

Learn about preparing data for ridge regression.

When ridge regression is enabled, different data preparation is likely to produce different results in terms of model coefficients and diagnostics. Oracle recommends that you enable Automatic Data Preparation for Generalized Linear Model models, especially when ridge regression is used.

Related Topics

- Data Preparation for GLM
  Learn about preparing data for the Generalized Linear Model (GLM) algorithm.

20.3 Scalable Feature Selection

Oracle Machine Learning for SQL supports a highly scalable and automated version of feature selection and generation for the Generalized Linear Model algorithm.

This scalable and automated capability can enhance the performance of the algorithm and improve accuracy and interpretability. Feature selection and generation are available for both linear regression and binary logistic regression.

20.3.1 Feature Selection

Feature selection is the process of choosing the terms to be included in the model. The fewer terms in the model, the easier it is for human beings to interpret its meaning. In addition, some columns may not be relevant to the value that the model is trying to predict. Removing such columns can enhance model accuracy.

20.3.1.1 Configuring Feature Selection

Feature selection is a build setting for Generalized Linear Model models. It is not enabled by default. When configured for feature selection, the algorithm automatically determines appropriate default behavior, but the following configuration options are available:

- The feature selection criteria can be AIC, SBIC, RIC, or α-investing. When the feature selection criteria is α-investing, feature acceptance can be either strict or relaxed.
- The maximum number of features can be specified.
- Features can be pruned in the final model. Pruning is based on t-statistics for linear regression or wald statistics for logistic regression.

20.3.1.2 Feature Selection and Ridge Regression

Feature selection and ridge regression are mutually exclusive. When feature selection is enabled, the algorithm can not use ridge.

> **Note:**
>
> If you configure the model to use both feature selection and ridge regression, then you get an error.

20.3.2 Feature Generation

Feature generation is the process of adding transformations of terms into the model. Feature generation enhances the power of models to fit more complex relationships between target and predictors.

20.3.2.1 Configuring Feature Generation

Learn about configuring feature generation.

Feature generation is only possible when feature selection is enabled. Feature generation is a build setting. By default, feature generation is not enabled.

The feature generation method can be either quadratic or cubic. By default, the algorithm chooses the appropriate method. You can also explicitly specify the feature generation method.

The following options for feature selection also affect feature generation:

- Maximum number of features
- Model pruning

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

20.4 Tuning and Diagnostics for GLM

The process of developing a Generalized Linear Model model typically involves a number of model builds. Each build generates many statistics that you can evaluate to determine the quality of your model. Depending on these diagnostics, you may want to try changing the model settings or making other modifications.

20.4.1 Build Settings

Specify the build settings for Generalized Linear Model (GLM).

You can use specify build settings.

Additional build settings are available to:

- Control the use of ridge regression.
- Specify the handling of missing values in the training data.
- Specify the target value to be used as a reference in a logistic regression model.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: Generalized Linear Models for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- Ridge Regression
  Understand the use of ridge regression for singularity (exact multicollinearity) in data.
- Data Preparation for GLM
  Learn about preparing data for the Generalized Linear Model (GLM) algorithm.
- Logistic Regression

20.4.2 Diagnostics

A Generalized Linear Model model generates many metrics to help you evaluate the quality of the model.

20.4.2.1 Coefficient Statistics

Learn about coeffficient statistics for linear and logistic regression.

The same set of statistics is returned for both linear and logistic regression, but statistics that do not apply to the machine learning technique are returned as NULL.

Coefficient statistics are returned by the model detail views for a Generalized Linear Model (GLM) model.

Related Topics

- Coefficient Statistics for Linear Regression
- Coefficient Statistics for Logistic Regression
- *Oracle Machine Learning for SQL User's Guide*

20.4.2.2 Global Model Statistics

Learn about high-level statistics describing the model.

Separate high-level statistics describing the model as a whole, are returned for linear and logistic regression. When ridge regression is enabled, fewer global details are returned.

Global statistics are returned by the model detail views for a Generalized Linear Model model.

Related Topics

- Global Model Statistics for Linear Regression
- Global Model Statistics for Logistic Regression
- Ridge Regression
  Understand the use of ridge regression for singularity (exact multicollinearity) in data.
- *Oracle Machine Learning for SQL User's Guide*

20.4.2.3 Row Diagnostics

Generate row-statistics by configuring the Generalized Linear Model (GLM) algorithm.

GLM generates per-row statistics if you specify the name of a diagnostics table in the build setting `GLMS_DIAGNOSTICS_TABLE_NAME`.

GLM requires a case ID to generate row diagnostics. If you provide the name of a diagnostic table but the data does not include a case ID column, an exception is raised.

Related Topics

- Row Diagnostics for Linear Regression
- Row Diagnostics for Logistic Regression

20.5 GLM Solvers

Generalized Linear Model (GLM) algorithm applies different solvers. These solvers employ different approaches for optimization.

The GLM algorithm supports four different solvers: Cholesky, QR, Stochastic Gradient Descent (SGD),and Alternating Direction Method of Multipliers (ADMM) (on top of L-BFGS). The Cholesky and QR solvers employ classical decomposition approaches. The Cholesky solver is faster compared to the QR solver but less stable numerically. The QR solver handles better rank deficient problems without the help of regularization.

The SGD and ADMM (on top of L-BFGS) solvers are best suited for large scale data. The SGD solver employs the stochastic gradient descent optimization algorithm while ADMM (on top of L-BFGS) uses the Broyden-Fletcher-Goldfarb-Shanno optimization algorithm within an Alternating Direction Method of Multipliers framework. The SGD solver is fast but is sensitive to parameters and requires suitable scaled data to achieve good convergence. The L-BFGS algorithm solves unconstrained optimization problems and is more stable and robust than SGD. Also, L-BFGS uses ADMM in conjunction, which, results in an efficient distributed optimization approach with low communication cost.

Related Topics

- DBMS_DATA_MINING - Algorithm Settings: Neural Network
- DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Models
- DBMS_DATA_MINING — Algorithm Settings: ADMM
- DBMS_DATA_MINING — Algorithm Settings: LBFGS

20.6 Data Preparation for GLM

Learn about preparing data for the Generalized Linear Model (GLM) algorithm.

Automatic Data Preparation (ADP) implements suitable data transformations for both linear and logistic regression.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: Generalized Linear Models for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting. Oracle recommends that you use ADP with GLM.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

20.6.1 Data Preparation for Linear Regression

Learn about Automatic Data Preparation (ADP) for the Generalized Linear Model (GLM) algorithm.

When ADP is enabled, the algorithm chooses a transformation based on input data properties and other settings. The transformation can include one or more of the following for numerical data: subtracting the mean, scaling by the standard deviation, or performing a correlation transformation (Neter, et. al, 1990). If the correlation transformation is applied to numeric data, it is also applied to categorical attributes.

Prior to standardization, categorical attributes are exploded into N-1 columns where N is the attribute cardinality. The most frequent value (mode) is omitted during the explosion transformation. In the case of highest frequency ties, the attribute values are sorted alpha-numerically in ascending order, and the first value on the list is omitted during the explosion. This explosion transformation occurs whether or not ADP is enabled.

In the case of high cardinality categorical attributes, the described transformations (explosion followed by standardization) can increase the build data size because the resulting data representation is dense. To reduce memory, disk space, and processing requirements, use an alternative approach. Under these circumstances, the VIF statistic must be used with caution.

Related Topics

- Ridge and Data Preparation
  Learn about preparing data for ridge regression.

> **See Also:**
>
> - Neter, J., Wasserman, W., and Kutner, M.H., "Applied Statistical Models", Richard D. Irwin, Inc., Burr Ridge, IL, 1990.

20.6.2 Data Preparation for Logistic Regression

Categorical attributes are exploded into *N*-1 columns where *N* is the attribute cardinality. The most frequent value (mode) is omitted during the explosion transformation. In the case of highest frequency ties, the attribute values are sorted alpha-numerically in ascending order and the first value on the list is omitted during the explosion. This explosion transformation occurs whether or not Automatic Data Preparation (ADP) is enabled.

When ADP is enabled, numerical attributes are scaled by the standard deviation. This measure of variability is computed as the standard deviation per attribute with respect to the origin (not the mean) (Marquardt, 1980).

See Also:

Marquardt, D.W., "A Critique of Some Ridge Regression Methods: Comment", Journal of the American Statistical Association, Vol. 75, No. 369 , 1980, pp. 87-91.

20.6.3 Missing Values

When building or applying a model, Oracle Machine Learning for SQL automatically replaces missing values of numerical attributes with the mean and missing values of categorical attributes with the mode.

You can configure the Generalized Linear Model algorithm to override the default treatment of missing values. With the `ODMS_MISSING_VALUE_TREATMENT` setting, you can cause the algorithm to delete rows in the training data that have missing values instead of replacing them with the mean or the mode. However, when the model is applied, OML4SQL performs the usual mean/mode missing value replacement. As a result, it is possible that the statistics generated from scoring does not match the statistics generated from building the model.

If you want to delete rows with missing values in the scoring the model, you must perform the transformation explicitly. To make build and apply statistics match, you must remove the rows with NULLs from the scoring data before performing the apply operation. You can do this by creating a view.

```
CREATE VIEW viewname AS SELECT * from tablename
     WHERE column_name1 is NOT NULL
     AND   column_name2 is NOT NULL
     AND   column_name3 is NOT NULL .....
```

Note:

In OML4SQL, missing values in nested data indicate sparsity, not values missing at random.

The value `ODMS_MISSING_VALUE_DELETE_ROW` is only valid for tables without nested columns. If this value is used with nested data, an exception is raised.

20.7 Linear Regression

Oracle Machine Learning for SQL supports linear regression as the Generalized Linear Model regression algorithm. The algorithm assumes no target transformation and constant variance over the range of target values. The algorithm uses the identity link function.

20.7.1 Poisson and Variance Link Function

The Poisson distribution is the number of occurrences of the event in a given time interval. It is a count distribution when the variable of interest is a discrete count variable.

For example, how many times per month will a grocery product be purchased? How many phone calls will be made per hour on the network? The predictors are the conditions that affect the average number events. The link function is in the following form:

$g(\mu) = \ln\mu = \beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+...\beta_n x_n$

Where average event count is μ.

The variance function is in the following form:

$Var(\mu)=\mu$

20.7.2 Negative Binomial Link Function and Variance

In Poisson distribution the variance is equal to the mean, however, sometimes, the variance of the predicted mean is larger than the mean. This occurrence in count data analysis is called **overdispersion**. Because the consequences are potentially so severe, models such as negative binomial regression can be applied.

The link function is in the following form:

$g(\mu) = \ln\mu = \beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+...\beta_n x_n$

Where average event count is μ.

20.7.3 Coefficient Statistics for Linear Regression

Generalized Linear Model regression models generate the following coefficient statistics:

- Linear coefficient estimate
- Standard error of the coefficient estimate
- t-value of the coefficient estimate
- Probability of the t-value
- Variance Inflation Factor (VIF)
- Standardized estimate of the coefficient
- Lower and upper confidence bounds of the coefficient

20.7.4 Global Model Statistics for Linear Regression

Generalized Linear Model regression models generate the following statistics that describe the model as a whole:

- Model degrees of freedom
- Model sum of squares
- Model mean square
- Model *F* statistic

- Model *F* value probability
- Error degrees of freedom
- Error sum of squares
- Error mean square
- Corrected total degrees of freedom
- Corrected total sum of squares
- Root mean square error
- Dependent mean
- Coefficient of variation
- R-Square
- Adjusted R-Square
- Akaike's information criterion
- Schwarz's Baysian information criterion
- Estimated mean square error of the prediction
- Hocking Sp statistic
- JP statistic (the final prediction error)
- Number of parameters (the number of coefficients, including the intercept)
- Number of rows
- Whether or not the model converged
- Whether or not a covariance matrix was computed

20.7.5 Row Diagnostics for Linear Regression

For linear regression, the diagnostics table has the columns described in the following table. All the columns are `NUMBER`, except the `CASE_ID` column, which preserves the type from the training data.

Table 20-1 Diagnostics Table for GLM Regression Models

Column	Description
`CASE_ID`	Value of the case ID column
`TARGET_VALUE`	Value of the target column
`PREDICTED_VALUE`	Value predicted by the model for the target
`HAT`	Value of the diagonal element of the hat matrix
`RESIDUAL`	Measure of error
`STD_ERR_RESIDUAL`	Standard error of the residual
`STUDENTIZED_RESIDUAL`	Studentized residual
`PRED_RES`	Predicted residual
`COOKS_D`	Cook's D influence statistic

20.8 Logistic Regression

Oracle Machine Learning for SQL supports binary logistic regression as a Generalized Linear Model classification algorithm. Link and variance functions are the mechanism that allows GLM to handle targets of a regression that departs in known ways from normality. In logistic regression, a link function is used to relate the explanatory variables (covariates) and the expectation of the response variable. Binomial regression predicts the probability of a success by applying the inverse of a specified link function to a linear combination of covariates. The specified inverse link function can be any monotonically increasing function that maps values from the range (-∞, ∞) to [0,1]. The inverse link function is created from cumulative distribution functions (CDFs) of well-known random distributions. The variance has a known functional relationship with the probability, and a binary target probability varies between zero and one. For logistic regression, the variance function is fixed to its known functional relationship with probability. However, there are other options for the link function. The link function not only transforms the target range into a linear-methods-friendly format, but it also represents a target concept. The analyst can use the target concept to interpret a forecast on two scales: the link scale and the transformed scale. The transformed scale in logistic regression is probability.

20.8.1 Logit Link Function

The logit link transforms a probability into the log of the odds ratio. The odds ratio is the ratio of the predicted probability of the positive to the predicted probability of the negative class. The log of the odds ratio has the appropriate range.

The odds ratio is a measure of the evidence for or against the positive target class. Odds ratios can be associated with particular predictor value. Odds ratios are naturally multiplicative, which makes the log of odds ratios additive. The log-odds ratio interprets the influence of a predictor as additive evidence for or against the positive class.

An advantage of the logit link is that the training data can be sampled independently from the two classes. This can be very significant in cases in which one class is rare or costly, such as the instances of a disease. Analysis of disease factors can be done directly from a sample of healthy people and a sample of people with the disease. This type of sampling is known as retrospective sampling.

For logistic regression, the logit link is the default. For technical reasons, this link is called the canonical link.

20.8.2 Probit Link Function

One approach to transforming the range of a probability to the range minus infinity to infinity is to choose a probability distribution that is defined on that range and assign the distribution value that corresponds to the probability as the target value.

For example, the probabilities, 0, 0.5 and 1.0 corresponds to the value -infinity, 0 and infinity in a standard normal distribution. An inverse cumulative distribution function is a function that determines the value that corresponds to a probability. In this approach, a user matches the particular probability distribution to assumptions regarding the distribution of the target. Users often find transformation of a target using the target's known associated distribution as natural.The probit link takes this approach, using the standard normal distribution. An example use case is an analysis of high blood pressure. Blood pressure is assumed to have a normal distribution.

20.8.3 Cloglog Link Function

The Complimentary Log-Log (cloglog) link is another example of using an inverse cumulative distribution function to transform the target. It differs from logit and probit function because it is asymmetric. It works best when the chance of an event is extremely low or extremely high.

Gumbel described these extreme value distributions. The cloglog model is closely related to continuous-time models for event occurrence. The cloglog link function corresponds to Gumbel CDF. The precipitation from the worst rainstorm in 100 years is an example of data that follows an extreme value distribution (the hundred year rain).

20.8.4 Cauchit Link Function

The Cauchit link is another application of an inverse cumulative distribution function to transform the target. In this case, the distribution is the Cauchy distribution. The Cauchy distribution is symmetric, however, it has infinite variance. An infinite variance means the probability decays slowly as the values become more extreme.

Such distributions are called fat-tailed. The Cauchit link is often used where fewer assumptions are justified with respect to the distribution of the target. The Cauchit link is used to measure data in binomial form when the variance is not considered to be finite.

20.8.5 Reference Class

You can use the build setting `GLMS_REFERENCE_CLASS_NAME` to specify the target value to be used as a reference in a binary logistic regression model. Probabilities are produced for the other (non-reference) class. By default, the algorithm chooses the value with the highest prevalence. If there are ties, the attributes are sorted alpha-numerically in an ascending order.

20.8.6 Class Weights

You can use the build setting `CLAS_WEIGHTS_TABLE_NAME` to specify the name of a class weights table. Class weights influence the weighting of target classes during the model build.

20.8.7 Coefficient Statistics for Logistic Regression

Generalized Linear Model classification models generate the following coefficient statistics:

- Name of the predictor
- Coefficient estimate
- Standard error of the coefficient estimate
- Wald chi-square value of the coefficient estimate
- Probability of the Wald chi-square value
- Standardized estimate of the coefficient
- Lower and upper confidence bounds of the coefficient

- Exponentiated coefficient
- Exponentiated coefficient for the upper and lower confidence bounds of the coefficient

20.8.8 Global Model Statistics for Logistic Regression

Generalized Linear Model classification models generate the following statistics that describe the model as a whole:

- Akaike's criterion for the fit of the intercept only model
- Akaike's criterion for the fit of the intercept and the covariates (predictors) model
- Schwarz's criterion for the fit of the intercept only model
- Schwarz's criterion for the fit of the intercept and the covariates (predictors) model
- -2 log likelihood of the intercept only model
- -2 log likelihood of the model
- Likelihood ratio degrees of freedom
- Likelihood ratio chi-square probability value
- Pseudo R-square Cox an Snell
- Pseudo R-square Nagelkerke
- Dependent mean
- Percent of correct predictions
- Percent of incorrect predictions
- Percent of ties (probability for two cases is the same)
- Number of parameters (the number of coefficients, including the intercept)
- Number of rows
- Whether or not the model converged
- Whether or not a covariance matrix was computed.

20.8.9 Row Diagnostics for Logistic Regression

For logistic regression, the diagnostics table has the columns described in the following table. All the columns are `NUMBER`, except the `CASE_ID` and `TARGET_VALUE` columns, which preserve the type from the training data.

Table 20-2 Row Diagnostics Table for Logistic Regression

Column	Description
`CASE_ID`	Value of the case ID column
`TARGET_VALUE`	Value of the target value
`TARGET_VALUE_PROB`	Probability associated with the target value
`HAT`	Value of the diagonal element of the hat matrix
`WORKING_RESIDUAL`	Residual with respect to the adjusted dependent variable
`PEARSON_RESIDUAL`	The raw residual scaled by the estimated standard deviation of the target

Table 20-2 (Cont.) Row Diagnostics Table for Logistic Regression

Column	Description
DEVIANCE_RESIDUAL	Contribution to the overall goodness of fit of the model
C	Confidence interval displacement diagnostic
CBAR	Confidence interval displacement diagnostic
DIFDEV	Change in the deviance due to deleting an individual observation
DIFCHISQ	Change in the Pearson chi-square

21
k-Means

Oracle Machine Learning for SQL supports enhanced *k*-Means clustering algorithm. Learn how to use the algorithm.

Related Topics

- Clustering Algorithms
  Learn different clustering algorithms used in Oracle Machine Learning for SQL.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: k-Means
- Automatic Data Preparation
- Model Detail Views for k-Means
- OML4SQL Examples
- OML4R k-Means Example
- OML4R GitHub Examples

21.1 About *k*-Means

The *k*-Means algorithm is a distance-based clustering algorithm that partitions the data into a specified number of clusters.

Distance-based algorithms rely on a distance function to measure the similarity between cases. Cases are assigned to the nearest cluster according to the distance function used.

21.1.1 Oracle Machine Learning for SQL Enhanced *k*-Means

Implementation of *k*-Means in Oracle Machine Learning for SQL.

OML4SQL implements an enhanced version of the *k*-Means algorithm with the following features:

- **Distance function**: The algorithm supports Euclidean and Cosine distance functions. The default is Euclidean.
- **Scalable Parallel Model build**: The algorithm uses a very efficient method of initialization based on *Bahmani, Bahman, et al. "Scalable k-means++." Proceedings of the VLDB Endowment 5.7 (2012): 622-633*.
- **Cluster properties**: For each cluster, the algorithm returns the centroid, a histogram for each attribute, and a rule describing the hyperbox that encloses the majority of the data

assigned to the cluster. The centroid reports the mode for categorical attributes and the mean and variance for numerical attributes.

This approach to *k*-Means avoids the need for building multiple *k*-Means models and provides clustering results that are consistently superior to the traditional *k*-Means.

21.1.2 Centroid

Defines a centroid in a cluster.

The **centroid** represents the most typical case in a cluster. For example, in a data set of customer ages and incomes, the centroid of each cluster would be a customer of average age and average income in that cluster. The centroid is a prototype. It does not necessarily describe any given case assigned to the cluster.

The attribute values for the centroid are the mean of the numerical attributes and the mode of the categorical attributes.

21.2 *k*-Means Algorithm Configuration

Learn about configuring the *k*-Means algorithm.

The Oracle Machine Learning for SQL enhanced *k*-Means algorithm supports several build-time settings. All the settings have default values. There is no reason to override the defaults unless you want to influence the behavior of the algorithm in some specific way.

You can configure *k*-Means by specifying the following considerations:

- Number of clusters
- Distance Function. The default distance function is Euclidean.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: k-Means for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

21.3 Data Preparation for *k*-Means

Learn about preparing data for *k*-Means algorithm.

Normalization is typically required by the *k*-Means algorithm. Automatic Data Preparation performs normalization for *k*-Means. If you do not use ADP, you must normalize numeric attributes before creating or applying the model.

When there are missing values in columns with simple data types (not nested), *k*-Means interprets them as missing at random. The algorithm replaces missing categorical values with the mode and missing numerical values with the mean.

When there are missing values in nested columns, *k*-Means interprets them as sparse. The algorithm replaces sparse numerical data with zeros and sparse categorical data with zero vectors.

Data can be constrained in a window size of 6 standard-deviations around the mean value by using the `KMNS_WINSORIZE` parameter. The `KMNS_WINSORIZE` parameter can be used whether ADP is set to `ON` or `OFF`. Values outside the range are mapped to the range's ends. This parameter is applicable only when the Euclidean distance is used.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- Prepare the Data

22
Minimum Description Length

Learn how to use Minimum Description Length, the supervised technique for calculating attribute importance.

- About MDL
- Data Preparation for MDL

Related Topics

- Feature Selection
  Learn how to perform feature selection and attribute importance.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Automatic Data Preparation
- Model Detail Views for Minimum Description Length
- OML4SQL Examples
- OML4R GitHub Examples

22.1 About MDL

Minimum Description Length (MDL) is an information theoretic model selection principle.

Information theoretic model selection principle is an important concept in information theory (the study of the quantification of information) and in learning theory (the study of the capacity for generalization based on empirical data).

MDL assumes that the simplest, most compact representation of the data is the best and most probable explanation of the data. The MDL principle is used to build Oracle Machine Learning for SQL attribute importance models.

The build process for attribute importance supports parallel execution.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*

22.1.1 Compression and Entropy

Data compression is the process of encoding information using fewer **bits** than what the original representation uses. The MDL Principle is based on the notion that the shortest description of the data is the most probable. In typical instantiations of this principle, a model is used to compress the data by reducing the uncertainty (entropy) as discussed below. The description of the data includes a description of the model and the data as described by the model.

Entropy is a measure of uncertainty. It quantifies the uncertainty in a random variable as the information required to specify its value. **Information** in this sense is defined as the number of yes/no questions known as **bits** (encoded as 0 or 1) that must be answered for a complete specification. Thus, the information depends upon the number of values that variable can assume.

For example, if the variable represents the sex of an individual, then the number of possible values is two: female and male. If the variable represents the salary of individuals expressed in whole dollar amounts, then the values can be in the range $0-$10B, or billions of unique values. Clearly it takes more information to specify an exact salary than to specify an individual's sex.

22.1.1.1 Values of a Random Variable: Statistical Distribution

Information (the number of bits) depends on the statistical distribution of the values of the variable as well as the number of values of the variable. If we are judicious in the choice of Yes/No questions, then the amount of information for salary specification cannot be as much as it first appears. Most people do not have billion dollar salaries. If most people have salaries in the range $32000-$64000, then most of the time, it requires only 15 questions to discover their salary, rather than the 30 required, if every salary from $0-$1000000000 were equally likely. In the former example, if the persons were known to be pregnant, then their sex is known to be female. There is no uncertainty, no Yes/No questions need be asked. The entropy is 0.

22.1.1.2 Values of a Random Variable: Significant Predictors

Suppose that for some random variable there is a predictor that when its values are known reduces the uncertainty of the random variable. For example, knowing whether a person is pregnant or not, reduces the uncertainty of the random variable sex-of-individual. This predictor seems like a valuable feature to include in a model. How about name? Imagine that if you knew the name of the person, you would also know the person's sex. If so, the name predictor would seemingly reduce the uncertainty to zero. However, if names are unique, then what was gained? Is the person named Sally? Is the person named George?... We would have as many Yes/No predictors in the name model as there are people. Therefore, specifying the name model would require as many bits as specifying the sex of each person.

22.1.1.3 Total Entropy

For a random variable, X, the **total entropy** is defined as minus the Probability(X) multiplied by the log to the base 2 of the Probability(X). This can be shown to be the variable's most efficient encoding.

22.1.2 Model Size

A Minimum Description Length (MDL) model takes into consideration the size of the model as well as the reduction in uncertainty due to using the model. Both model size and entropy are measured in bits. For our purposes, both numeric and categorical predictors are binned. Thus the size of each single predictor model is the number of predictor bins. The uncertainty is reduced to the within-bin target distribution.

22.1.3 Model Selection

Minimum Description Length (MDL) considers each attribute as a simple predictive model of the target class. **Model selection** refers to the process of comparing and ranking the single-predictor models.

MDL uses a communication model for solving the model selection problem. In the communication model there is a sender, a receiver, and data to be transmitted.

These single predictor models are compared and ranked with respect to the MDL metric, which is the relative compression in bits. MDL penalizes model complexity to avoid over-fit. It is a principled approach that takes into account the complexity of the predictors (as models) to make the comparisons fair.

22.1.4 The MDL Metric

Attribute importance uses a two-part code as the metric for transmitting each unit of data. The first part (preamble) transmits the model. The parameters of the model are the target probabilities associated with each value of the prediction.

For a target with *j* values and a predictor with *k* values, n_i (i= 1,..., k) rows per value, there are C_i, the combination of *j*-1 things taken n_i-1 at a time possible conditional probabilities. The size of the preamble in bits can be shown to be Sum($\log_2(C_i)$), where the sum is taken over *k*. Computations like this represent the penalties associated with each single prediction model. The second part of the code transmits the target values using the model.

It is well known that the most compact encoding of a sequence is the encoding that best matches the probability of the symbols (target class values). Thus, the model that assigns the highest probability to the sequence has the smallest target class value transmission cost. In bits, this is the Sum($\log_2(p_i)$), where the p_i are the predicted probabilities for row $_i$ associated with the model.

The predictor rank is the position in the list of associated description lengths, smallest first.

22.2 Data Preparation for MDL

Learn about preparing data for Minimum Description Length (MDL).

Automatic Data Preparation performs supervised binning for MDL. Supervised binning uses decision trees to create the optimal bin boundaries. Both categorical and numerical attributes are binned.

MDL handles missing values naturally as missing at random. The algorithm replaces sparse numerical data with zeros and sparse categorical data with zero vectors. Missing values in nested columns are interpreted as sparse. Missing values in columns with simple data types are interpreted as missing at random.

If you choose to manage your own data preparation, keep in mind that MDL usually benefits from binning. However, the discriminating power of an attribute importance model can be significantly reduced when there are outliers in the data and external equal-width binning is used. This technique can cause most of the data to concentrate in a few bins (a single bin in extreme cases). In this case, quantile binning is a better solution.

> **See Also:**
>
> DBMS_DATA_MINING — Automatic Data Preparation for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- Prepare the Data

23 Multivariate State Estimation Technique - Sequential Probability Ratio Test

The Multivariate State Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT) algorithm monitors critical processes and detects subtle anomalies.

- About Multivariate State Estimation Technique - Sequential Probability Ratio Test
- Score an MSET-SPRT Model

Related Topics

- Anomaly Detection
  Learn how to detect rare cases in the data through anomaly detection - an unsupervised function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test
- Automatic Data Preparation
- Model Detail View for Multivariate State Estimation Technique - Sequential Probability Ratio Test
- OML4SQL Examples
- OML4R GitHub Examples

23.1 About Multivariate State Estimation Technique - Sequential Probability Ratio Test

Multivariate state Estimation Technique - Sequential Probability Ratio Test (MSET-SPRT) is an algorithm for anomaly detection and statistical testing.

MSET is a nonlinear, nonparametric anomaly detection machine learning technique that calibrates the expected behavior of a system based on historical data from the normal operational sequence of monitored signals. It incorporates the learned behavior of a system into a persistent model that represents the normal estimated behavior. You can deploy the model to evaluate a subsequent stream of live signal vectors using Oracle Machine Learning for SQL scoring functions. To form a hypothesis as to the overall health of the system, these functions calculate the difference between the estimated and the actual signal values (residuals) and use SPRT calculations to determine whether any of the signals have become degraded.

To build a good model, MSET requires sufficient historical data that adequately captures all normal modes of behavior of the system. Incomplete data results in false alerts when the system enters a mode of operation that was poorly represented in the historical data. MSET assumes that the characteristics of the data being monitored do not change over time. Once deployed, MSET is a stationary model and does not evolve as it monitors a data stream.

Both MSET and SPRT operate on continuous time-ordered sensor data. If the raw data stream needs to be pre-processed or sampled, you must do that before you pass the data to the MSET-SPRT model.

The `ALGO_MSET_SPRT` algorithm is designated as a classification machine learning technique. It generates a model in which each data row is labeled as either normal or anomalous. For anomalous predictions, the prediction details provide a list of the sensors that show the anomaly and a weight.

When creating an MSET-SPRT model with the `DBMS_DATA_MINING.CREATE_MODEL` function, use the `case_id` argument to provide a unique row identifier for the time-ordered data that the algorithm requires. The build is then able to sort the training data and create windows for sampling and variance estimation. If you do not provide a `case_id`, then an exception occurs.

MSET-SPRT supports only numeric data. An exception occurs if other column types are in the build data.

When the number of sensors is very high, MSET-SPRT leverages random projections to improve the scalability and robustness of the algorithm. Random projections is a technique that reduces dimensionality while preserving pairwise distances. By randomly projecting the sensor data, the problem is solved in a distance-preserving, lower-dimension space. The MSET hypothesis testing approach is applied on the projected data where each random projection can be viewed as a Monte Carlo simulation of system health. The overall probability of an anomaly follows a binomial distribution with the number of projections as the number of trials and the number of alerting projections as the number of successes.

> **Note:**
>
> An MSET-SPRT model with random projections does not produce prediction details. When random projections are employed, the nature of the prediction output changes. The prediction captures the global health of the system and it is not possible to attribute the cause to individual attributes. Therefore, `PREDICTION_DETAILS` returns an empty list.

> **See Also:**
>
> DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test

23.2 Score an MSET-SPRT Model

Scoring data with MSET-SPRT models is similar to scoring with classification algorithms, except that the SPRT methodology relies on ordered data because it tracks gradual shifts over multiple MSET predictions.

This is different than the typical usage of Oracle Database SQL prediction functions, which do not keep state information between rows.

The following functions are supported: `PREDICTION`, `PREDICTION_COST`, `PREDICTION_DETAILS`, `PREDICTION_PROBABILITY`, and `PREDICTION_SET`. These functions have syntax new in Oracle Database 21c for scoring MSET-SPRT models. That syntax has an `ORDER BY` clause to order and window the historical data.

The prediction functions return the following information:

- `PREDICTION` indicates whether the record is flagged as anomalous. It uses the same automatically generated labels as one-class SVM models: 1 for normal and 0 for anomalous.
- `PREDICTION_COST` performs an auto-cost analysis or a user-specified cost. A user-specified cost typically assigns a higher cost to false positives than to false negatives.
- `PREDICTION_DETAILS` specify the signals that support the prediction along with a weight.
- `PREDICTION_PROBABILITY` conveys a measure of certainty based on the consolidation logic.
- `PREDICTION_SET` returns the set of predictions (0, 1) and the corresponding prediction probabilities for each observation.

> **Note:**
>
> If the values in one or more of the columns specified in the `ORDER BY` clause are not unique, or do not represent a true chronology of data sample values, the SPRT predictions are not guaranteed to be meaningful or consistent between query executions.

Unlike other classification models, an MSET-SPRT model has no obvious probability measure associated with the anomalous label for the record as a whole. However, the consolidation logic can produce a measure of uncertainty in place of probability. For example, if an alert is raised for 2 anomalies over a window of 5 observations, a certainty of 0.5 is reported when 2 anomalies are seen within the 5 observation window. The certainty increases if more than 3 anomalies are seen and decreases if no anomalies are seen.

The `PREDICTION_DETAILS` function accommodates output of varying forms and can convey the required information regarding the individual signals that triggered an alarm. When random projections are engaged, only the overall `PREDICTION` and `PREDICTION_PROBABILITY` are computed and `PREDICTION_DETAILS` are not reported.

You must score the historical data in order to tune the SPRT parameters, such as false alerts and miss rates or consolidation logic, before you deploy the MSET model. The SPRT parameters are embedded in the model object to facilitate deployment. While scoring in the database is needed for parameter tuning and forensic analysis on historical data, monitoring a stream of sensor data is more easily done outside of the database in an IoT service or on the edge device itself.

You can build and score an MSET-SPRT model as a partitioned model if the same columns that you use to build the model are present in the input scoring data set. If those columns are not present, the query results in an error.

Related Topics

- SQL Scoring Functions
- SQL Scoring Functions
- MSET_SPRT example on GitHub

24

Naive Bayes

Learn how to use the Naive Bayes classification algorithm.

- About Naive Bayes
- Tuning a Naive Bayes Model
- Data Preparation for Naive Bayes

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Naive Bayes
- Automatic Data Preparation
- Model Detail Views for Naive Bayes
- OML4SQL Examples
- OML4R Naive Bayes Example
- OML4R GitHub Examples

24.1 About Naive Bayes

Naive Bayes algorithm is based on conditional probabilities. It uses Bayes' theorem, a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data.

Bayes' theorem finds the probability of an event occurring given the probability of another event that has already occurred. If `B` represents the dependent event and `A` represents the prior event, Bayes' theorem can be stated as follows.

> **Note:**
>
> Prob(B given A) = Prob(A and B)/Prob(A)

To calculate the probability of `B` given `A`, the algorithm counts the number of cases where `A` and `B` occur together and divides it by the number of cases where `A` occurs alone.

Example 24-1 Use Bayes' Theorem to Predict an Increase in Spending

Suppose you want to determine the likelihood that a customer under 21 increases spending. In this case, the prior condition (`A`) is "under 21," and the dependent condition (`B`) is "increase spending."

If there are 100 customers in the training data and 25 of them are customers under 21 who have increased spending, then:

Prob(A and B) = 25%

If 75 of the 100 customers are under 21, then:

Prob(A) = 75%

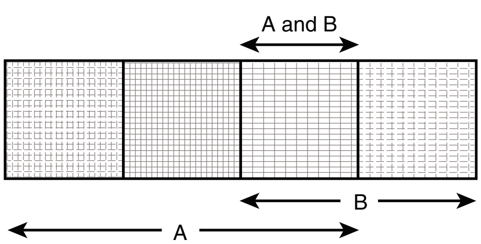
Bayes' theorem predicts that 33% of customers under 21 are likely to increase spending (25/75).

The cases where both conditions occur together are referred to as **pairwise**. In Example 24-1, 25% of all cases are pairwise.

The cases where only the prior event occurs are referred to as **singleton**. In Example 24-1, 75% of all cases are singleton.

A visual representation of the conditional relationships used in Bayes' theorem is shown in the following figure.

Figure 24-1 Conditional Probabilities in Bayes' Theorem



P(A) = 3/4
P(B) = 2/4
P(A and B) = P(AB) = 1/4
P(A|B) = P(AB) / P(B) = (1/4) / (2/4) = 1/2
P(B|A) = P(AB) / P(A) = (1/4) / (3/4) = 1/3

For purposes of illustration, Example 24-1 and Figure 24-1 show a dependent event based on a single independent event. In reality, the Naive Bayes algorithm must usually take many independent events into account. In Example 24-1, factors such as income, education, gender, and store location might be considered in addition to age.

Naive Bayes makes the assumption that each predictor is conditionally independent of the others. For a given target value, the distribution of each predictor is independent of the other predictors. In practice, this assumption of independence, even when violated, does not degrade the model's predictive accuracy significantly, and makes the difference between a fast, computationally feasible algorithm and an intractable one.

Sometimes the distribution of a given predictor is clearly not representative of the larger population. For example, there might be only a few customers under 21 in the training data, but in fact there are many customers in this age group in the wider customer base. To compensate for this, you can specify **prior probabilities** when training the model.

Related Topics

- Priors and Class Weights
  Learn about Priors and Class Weights in a classification model to produce a useful result.

24.1.1 Advantages of Naive Bayes

Learn about the advantages of Naive Bayes.

The Naive Bayes algorithm affords fast, highly scalable model building and scoring. It scales linearly with the number of predictors and rows.

The build process for Naive Bayes supports parallel execution. (Scoring supports parallel execution irrespective of the algorithm.)

Naive Bayes can be used for both binary and multiclass classification problems.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*

24.2 Tuning a Naive Bayes Model

Introduces about probability calculation of pairwise occurrences and percentage of singleton occurrences.

Naive Bayes calculates a probability by dividing the percentage of pairwise occurrences by the percentage of singleton occurrences. If these percentages are very small for a given predictor, they probably do not contribute to the effectiveness of the model. Occurrences below a certain threshold can usually be ignored.

The following build settings are available for adjusting the probability thresholds. You can specify:

- The minimum percentage of pairwise occurrences required for including a predictor in the model.
- The minimum percentage of singleton occurrences required for including a predictor in the model .

The default thresholds work well for most models, so you need not adjust these settings.

> **See Also:**
>
> DBMS_DATA_MINING — Algorithm Settings: Naive Bayes for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

24.3 Data Preparation for Naive Bayes

Learn about preparing the data for Naive Bayes.

Automatic Data Preparation (ADP) performs supervised binning for Naive Bayes. Supervised binning uses decision trees to create the optimal bin boundaries. Both categorical and numeric attributes are binned.

Naive Bayes handles missing values naturally as missing at random. The algorithm replaces sparse numerical data with zeros and sparse categorical data with zero vectors. Missing values in nested columns are interpreted as sparse. Missing values in columns with simple data types are interpreted as missing at random.

If you choose to manage your own data preparation, keep in mind that Naive Bayes usually requires binning. Naive Bayes relies on counting techniques to calculate probabilities. Columns must be binned to reduce the cardinality as appropriate. Numerical data can be binned into ranges of values (for example, low, medium, and high), and categorical data can be binned into meta-classes (for example, regions instead of cities). Equi-width binning is not recommended, since outliers cause most of the data to concentrate in a few bins, sometimes a single bin. As a result, the discriminating power of the algorithms is significantly reduced

Related Topics

- Prepare the Data

25 Neural Network

Learn about the Neural Network algorithms for regression and classification machine learning techniques.

- About Neural Network
- Data Preparation for Neural Network
- Neural Network Algorithm Configuration
- Scoring with Neural Network

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- Regression
  Learn how to predict a continuous numerical target through regression - the supervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: Neural Network
- Automatic Data Preparation
- Model Detail Views for Neural Network
- OML4SQL Examples
- OML4R Neural Network Example
- OML4R GitHub Examples

25.1 About Neural Network

The Neural Network algorithm in Oracle Machine Learning for SQL is designed for machine learning techniques like classification and regression.

In machine learning, an artificial neural network is an algorithm inspired from biological neural network and is used to estimate or approximate functions that depend on a large number of generally unknown inputs. An artificial neural network is composed of a large number of interconnected neurons which exchange messages between each other to solve specific problems. They learn by examples and tune the weights of the connections among the neurons during the learning process. The Neural Network algorithm is capable of solving a wide variety of tasks such as computer vision, speech recognition, and various complex business problems.

Related Topics

- About Regression
  Regression is an Oracle Machine Learning for SQL function that predicts numeric values along a continuum.

- About Classification
  Classification is a machine learning technique that assigns items in a collection to target categories or classes.

25.1.1 Neurons and Activation Functions

Neurons are the building blocks of a neural network.

A neuron takes one or more inputs having different weights and has an output which depends on the inputs. The output is achieved by adding up inputs of each neuron with weights and feeding the sum into the activation function.

A Sigmoid function is usually the most common choice for activation function but other non-linear functions, piecewise linear functions or step functions are also used. The Rectified Linear Units function `NNET_ACTIVATIONS_RELU` is a commonly used activation function that addresses the vanishing gradient problem for larger neural networks.

The following are some examples of activation functions:

- Logistic Sigmoid function
- Linear function
- Tanh function
- Arctan function
- Bipolar sigmoid function
- Rectified Linear Units

25.1.2 Loss or Cost function

A loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.

An optimization problem seeks to minimize a loss function. The form of loss function is chosen based on the nature of the problem and mathematical needs.

The following are the different loss functions for different scenarios:

- Binary classification: binary cross entropy loss function.
- Multi-class classification: multi cross entropy loss function.
- Regression: squared error function.

25.1.3 Forward-Backward Propagation

Understand forward-backward propagation.

Forward propagation computes the loss function value by weighted summing the previous layer neuron values and applying activation functions. Backward propagation calculates the gradient of a loss function with respect to all the weights in the network. The weights are initialized with a set of random numbers uniformly distributed within a region specified by user (by setting weights boundaries), or region defined by the number of nodes in the adjacent layers (data driven). The gradients are fed to an optimization method which in turn uses them to update the weights, in an attempt to minimize the loss function.

25.1.4 Optimization Solvers

An optimization solver is a function that searches for the optimal solution of the loss function to find the extreme value (maximum or minimum) of the loss (cost) function.

Oracle Machine Learning implements Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) together with line search and the Adam solver.

Limited-memory Broyden–Fletcher–Goldfarb–Shanno Solver

L-BFGS is a Quasi-Newton method. This method uses rank-one updates specified by gradient evaluations to approximate a Hessian matrix. This method only needs a limited amount of memory. L-BFGS is used to find the descent direction and line search is used to find the appropriate step size. The number of historical copies kept in the L-BFGS solver is defined by the `LBFGS_HISTORY_DEPTH` solver setting. When the number of iterations is smaller than the history depth, the Hessian computed by L-BFGS is accurate. When the number of iterations is larger than the history depth, the Hessian computed by L-BFGS is an approximation. Therefore, the history depth should not be too small or too large to avoid making the computation too slow. Typically, the value is between `3` and `10`.

Adam Solver

Adam is an extension to stochastic gradient descent that uses mini-batch optimization. The L-BFGS solver may be a more stable solver whereas the Adam solver can make progress faster by seeing less data. Adam is computationally efficient, with little memory requirements, and is well-suited for problems that are large in terms of data or parameters or both.

25.1.5 Regularization

Understand regularization.

Regularization refers to a process of introducing additional information to solve an ill-posed problem or to prevent over-fitting. Ill-posed or over-fitting can occur when a statistical model describes random errors or noise instead of the underlying relationship. Typical regularization techniques include L1-norm regularization, L2-norm regularization, and held-aside.

Held-aside is usually used for large training date sets whereas L1-norm regularization and L2-norm regularization are mostly used for small training date sets.

25.1.6 Convergence Check

This checks if the optimal solution has been reached and if the iterations of the optimization has come to an end.

In L-BFGS solver, the convergence criteria includes maximum number of iterations, infinity norm of gradient, and relative error tolerance. For held-aside regularization, the convergence criteria checks the loss function value of the test data set, as well as the best model learned so far. The training is terminated when the model becomes worse for a specific number of iterations (specified by `NNET_HELDASIDE_MAX_FAIL`), or the loss function is close to zero, or the relative error on test data is less than the tolerance.

25.1.7 LBFGS_SCALE_HESSIAN

Defines `LBFGS_SCALE_HESSIAN`.

It specifies how to set the initial approximation of the inverse Hessian at the beginning of each iteration. If the value is set to be `LBFGS_SCALE_HESSIAN_ENABLE`, then we approximate the initial inverse Hessian with Oren-Luenberger scaling. If it is set to be `LBFGS_SCALE_HESSIAN_DISABLE`, then we use identity as the approximation of the inverse Hessian at the beginning of each iteration.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

25.1.8 NNET_HELDASIDE_MAX_FAIL

Defines `NNET_HELDASIDE_MAX_FAIL`.

Validation data (held-aside) is used to stop training early if the network performance on the validation data fails to improve or remains the same for `NNET_HELDASIDE_MAX_FAIL` epochs in a row.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

25.2 Data Preparation for Neural Network

Learn about preparing data for the Neural Network algorithm.

The algorithm automatically "explodes" categorical data into a set of binary attributes, one per category value. Oracle Machine Learning for SQL algorithms automatically handle missing values and therefore, missing value treatment is not necessary.

The algorithm automatically replaces missing categorical values with the mode and missing numerical values with the mean. The algorithm requires the normalization of numeric input and it uses z-score normalization. The normalization occurs only for two-dimensional numeric columns (not nested). Normalization places the values of numeric attributes on the same scale and prevents attributes with a large original scale from biasing the solution. Neural Network scales the numeric values in nested columns by the maximum absolute value seen in the corresponding columns.

Related Topics

- Prepare the Data

25.3 Neural Network Algorithm Configuration

Configure the Neural Network algorithm.

Specify Nodes Per Layer

```
INSERT INTO SETTINGS_TABLE (setting_name, setting_value) VALUES
                   ('NNET_NODES_PER_LAYER', '2,3');
```

Specify Activation Functions Per Layer

`NNET_ACTIVATIONS` setting specifies the activation functions or hidden layers.

> **See Also:**
>
> DBMS_DATA_MINING —Algorithm Settings: Neural Network for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

25.4 Scoring with Neural Network

Learn to score with a Neural Network algorithm.

Scoring with Neural Network is the same as any other classification or regression algorithm. The following functions are supported: `PREDICTION`, `PREDICTION_PROBABILITY`, `PREDICTION_COST`, `PREDICTION_SET`, and `PREDICTION_DETAILS`.

Related Topics

- *Oracle Database SQL Language Reference*

26

Non-Negative Matrix Factorization

Learn how to use Non-Negative Matrix Factorization (NMF), an unsupervised algorithm, that Oracle Machine Learning for SQL uses for feature extraction.

- About NMF
- Tuning the NMF Algorithm
- Data Preparation for NMF

Related Topics

- Feature Extraction
  Learn how to perform attribute reduction using feature extraction as an unsupervised function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: Non-Negative Matrix Factorization
- Automatic Data Preparation
- Model Detail Views for Non-Negative Matrix Factorization
- OML4SQL Examples
- OML4R Non-Negative Matrix Factorization Example
- OML4R GitHub Examples

> **See Also:**
>
> Paper "Learning the Parts of Objects by Non-Negative Matrix Factorization" by D. D. Lee and H. S. Seung in *Nature* (401, pages 788-791, 1999)

26.1 About NMF

Non-Negative Matrix Factorization is useful when there are many attributes and the attributes are ambiguous or have weak predictability. By combining attributes, NMF can produce meaningful patterns, topics, or themes. NMF is a feature extraction algorithm.

Each feature created by NMF is a linear combination of the original attribute set. Each feature has a set of coefficients, which are a measure of the weight of each attribute on the feature. There is a separate coefficient for each numerical attribute and for each distinct value of each categorical attribute. The coefficients are all non-negative.

26.1.1 Matrix Factorization

Non-Negative Matrix Factorization uses techniques from multivariate analysis and linear algebra. It decomposes the data as a matrix *M* into the product of two lower ranking matrices *W* and *H*. The sub-matrix *W* contains the NMF basis; the sub-matrix *H* contains the associated coefficients (weights).

The algorithm iteratively modifies of the values of *W* and *H* so that their product approaches *M*. The technique preserves much of the structure of the original data and guarantees that both basis and weights are non-negative. The algorithm terminates when the approximation error converges or a specified number of iterations is reached.

The NMF algorithm must be initialized with a seed to indicate the starting point for the iterations. Because of the high dimensionality of the processing space and the fact that there is no global minimization algorithm, the appropriate initialization can be critical in obtaining meaningful results. Oracle Machine Learning for SQL uses a random seed that initializes the values of W and H based on a uniform distribution. This approach works well in most cases.

26.1.2 Scoring with NMF

Non-Negative Matrix Factorization (NMF) can be used as a pre-processing step for dimensionality reduction in classification, regression, clustering, and other machine learning tasks. Scoring an NMF model produces data projections in the new feature space. The magnitude of a projection indicates how strongly a record maps to a feature.

The SQL scoring functions for feature extraction support NMF models. When the functions are invoked with the analytical syntax, the functions build and apply a transient NMF model. The feature extraction functions are: `FEATURE_DETAILS`, `FEATURE_ID`, `FEATURE_SET`, and `FEATURE_VALUE`.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

26.1.3 Text Analysis with NMF

Learn about text analysis with Non-Negative Matrix Factorization (NMF).

NMF is especially well-suited for analyzing text. In a text document, the same word can occur in different places with different meanings. For example, "hike" can be applied to the outdoors or to interest rates. By combining attributes, NMF introduces context, which is essential for explanatory power:

- "hike" + "mountain" -> "outdoor sports"
- "hike" + "interest" -> "interest rates"

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

26.2 Tuning the NMF Algorithm

Learn about configuring parameters for Non-Negative Matrix Factorization (NMF).

Oracle Machine Learning for SQL supports five configurable parameters for NMF. All of them have default values which are appropriate for most applications of the algorithm. The NMF settings are:

- Number of features. By default, the number of features is determined by the algorithm.
- Convergence tolerance. The default is .05.
- Number of iterations. The default is 50.
- Random seed. The default is -1.
- Non-negative scoring. You can specify whether negative numbers must be allowed in scoring results. By default they are allowed.

See Also:

DBMS_DATA_MINING —Algorithm Settings: Non-Negative Matrix Factorization for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

26.3 Data Preparation for NMF

You can use Automatic Data Preparation (ADP) or supply your transformation like binning or normalization to prepare the data for Non-Negative Matrix Factorization (NMF).

ADP normalizes numerical attributes for NMF.

When there are missing values in columns with simple data types (not nested), NMF interprets them as missing at random. The algorithm replaces missing categorical values with the mode and missing numerical values with the mean.

When there are missing values in nested columns, NMF interprets them as sparse. The algorithm replaces sparse numerical data with zeros and sparse categorical data with zero vectors.

If you choose to manage your own data preparation, keep in mind that outliers can significantly impact NMF. Use a clipping transformation before binning or normalizing. NMF typically benefits from normalization. However, outliers with min-max normalization cause poor matrix factorization. To improve the matrix factorization, you need to decrease the error tolerance. This in turn leads to longer build times.

Related Topics

- Prepare the Data

27 O-Cluster

Learn how to use orthogonal partitioning clustering (O-Cluster), an Oracle-proprietary clustering algorithm.

Related Topics

- Clustering Algorithms
  Learn different clustering algorithms used in Oracle Machine Learning for SQL.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: O-Cluster
- Automatic Data Preparation
- Model Detail Views for O-Cluster
- OML4SQL Examples
- OML4R O-Cluster Example
- OML4R GitHub Examples

> **See Also:**
>
> Campos, M.M., Milenova, B.L., "Clustering Large Databases with Numeric and Nominal Values Using Orthogonal Projections", Oracle Data Mining Technologies, Oracle Corporation.

27.1 About O-Cluster

O-Cluster is a fast, scalable grid-based clustering algorithm well-suited for analysing large, high-dimensional data sets. The algorithm can produce high quality clusters without relying on user-defined parameters.

The objective of O-Cluster is to identify areas of high density in the data and separate the dense areas into clusters. It uses axis-parallel uni-dimensional (orthogonal) data projections to identify the areas of density. The algorithm looks for splitting points that result in distinct clusters that do not overlap and are balanced in size.

O-Cluster operates recursively by creating a binary tree hierarchy. The number of leaf clusters is determined automatically. The algorithm can be configured to limit the maximum number of clusters.

27.1.1 Partitioning Strategy

Partitioning strategy refers to the process of discovering areas of density in the attribute histograms. The process differs for numerical and categorical data. When both are present in the data, the algorithm performs the searches separately and then compares the results.

In choosing a partition, the algorithm balances two objectives: finding well separated clusters, and creating clusters that are balanced in size. The following paragraphs detail how partitions for numerical and categorical attributes are identified.

27.1.1.1 Partitioning Numerical Attributes

To find the best valid cutting plane, O-Cluster searches the attribute histograms for bins of low density (valleys) between bins of high density (peaks).

O-Cluster attempts to find a pair of peaks with a valley between them where the difference between the peak and valley histogram counts is statistically significant.

A **sensitivity** level parameter specifies the lowest density that may be considered a peak. Sensitivity is an optional parameter for numeric data. It may be used to filter the splitting point candidates.

27.1.1.2 Partitioning Categorical Attributes

Categorical values do not have an intrinsic order associated with them. Therefore it is impossible to apply the notion of histogram peaks and valleys that is used to partition numerical values. Instead the counts of individual values form a histogram.

Bins with large counts are interpreted as regions with high density. The clustering objective is to separate these high-density areas and effectively decrease the entropy (randomness) of the data.

O-Cluster identifies the histogram with highest entropy along the individual projections. Entropy is measured as the number of bins above **sensitivity** level. O-Cluster places the two largest bins into separate partitions, thereby creating a splitting predicate. The remainder of the bins are assigned randomly to the two resulting partitions.

27.1.2 Active Sampling

The O-Cluster algorithm operates on a data buffer of a limited size. It uses an active sampling mechanism to handle data sets that do not fit into memory.

After processing an initial random sample, O-Cluster identifies cases that are of no further interest. Such cases belong to *frozen* partitions where further splitting is highly unlikely. These cases are replaced with examples from *ambiguous* regions where further information (additional cases) is needed to find good splitting planes and continue partitioning. A partition is considered ambiguous if a valid split can only be found at a lower confidence level.

Cases associated with frozen partitions are marked for deletion from the buffer. They are replaced with cases belonging to ambiguous partitions. The histograms of the ambiguous partitions are updated and splitting points are reevaluated.

27.1.3 Process Flow

At a high level, O-Cluster algorithm evaluates, splits the data into new partition, and searches for cutting planes inside the new partitions.

The O-Cluster algorithm evaluates possible splitting points for all projections in a partition, selects the best one, and splits the data into two new partitions. The algorithm proceeds by searching for good cutting planes inside the newly created partitions. Thus, O-Cluster creates a binary tree structure that divides the input space into rectangular regions with no overlaps or gaps.

The main processing stages are:

1. Load the buffer. Assign all cases from the initial buffer to a single active root partition.
2. Compute histograms along the orthogonal uni-dimensional projections for each active partition.
3. Find the best splitting points for active partitions.
4. Flag ambiguous and frozen partitions.
5. When a valid separator exists, split the active partition into two new active partitions and start over at step 2.
6. Reload the buffer after all recursive partitioning on the current buffer is completed. Continue loading the buffer until either the buffer is filled again, or the end of the data set is reached, or until the number of cases is equal to the data buffer size.

> **Note:**
>
> O-Cluster requires at most one pass through the data

27.1.4 Scoring

The clusters discovered by O-Cluster are used to generate a Bayesian probability model that can be used to score new data.

The generated probability model is a mixture model where the mixture components are represented by a product of independent normal distributions for numerical attributes and multinomial distributions for categorical attributes.

27.2 Tuning the O-Cluster Algorithm

You can configure build-time settings for O-Cluster.

The O-Cluster algorithm supports two build-time settings. Both settings have default values. There is no reason to override the defaults unless you want to influence the behavior of the algorithm in some specific way.

You can configure O-Cluster by specifying the following:

Sensitivity factor — A fraction that specifies the peak density required for separating a new cluster.

See Also:

DBMS_DATA_MINING — Algorithm Settings: O-Cluster for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

Related Topics

- Active Sampling
  The O-Cluster algorithm operates on a data buffer of a limited size. It uses an active sampling mechanism to handle data sets that do not fit into memory.
- Partitioning Strategy

27.3 Data Preparation for O-Cluster

Use Automatic Data Preparation (ADP) to prepare the data for O-Cluster.

ADP bins numerical attributes for O-Cluster. It uses a specialized form of equi-width binning that computes the number of bins per attribute automatically. Numerical columns with all nulls or a single value are removed. O-Cluster handles missing values naturally as missing at random.

Note:

O-Cluster does not support nested columns, sparse data, or unstructured text.

Related Topics

- Prepare the Data

27.3.1 User-Specified Data Preparation for O-Cluster

You can prepare the data for O-Cluster by considering the points listed here.

Keep the following in mind if you choose to prepare the data for O-Cluster:

- O-Cluster does not necessarily use all the input data when it builds a model. It reads the data in batches (the default batch size is 50000). It only reads another batch if it believes, based on statistical tests, that uncovered clusters can still exist.
- Binary attributes must be declared as categorical.
- Automatic equi-width binning is highly recommended. The bin identifiers are expected to be positive consecutive integers starting at 1.
- The presence of outliers can significantly impact clustering algorithms. Use a clipping transformation before binning or normalizing. Outliers with equi-width

binning can prevent O-Cluster from detecting clusters. As a result, the whole population appears to fall within a single cluster.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

28
R Extensibility

This topic applies only to Oracle on-premises.

Learn how to build an analytics model and score in R. The R extensible algorithms are enhanced to support and register additional algorithms for users who use SQL and graphical user interface.

- Oracle Machine Learning for SQL with R Extensibility
- Scoring with R
- About Algorithm Metadata Registration

Related Topics

- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: ALGO_EXTENSIBLE_LANG
- OML4SQL Examples
- OML4R Extensible R Example
- OML4R GitHub Examples

28.1 Oracle Machine Learning for SQL with R Extensibility

Learn how you can use Oracle Machine Learning for SQL to build, score, and view machine learning models as well as R models.

The OML4SQL framework is enhanced extending the OML4SQL algorithm set with algorithms from the open source R ecosystem. Oracle Machine Learning for SQL is implemented in the Oracle Database kernel. The OML4SQL models are Database schema objects. With the extensibility enhancement, the OML4SQL framework can build, score, and view both OML4SQL models and R models.

Registration of R scripts

The R engine on the database server runs the R scripts to build, score, and view R models. These R scripts must be registered with the database beforehand by a privileged user with `rqAdmin` role. You must first install Oracle Machine Learning for R to register the R scripts.

Functions of Oracle Machine Learning for SQL with R Model

The following functions are supported for an R model:

- OML4SQL `DBMS_DATA_MINING` package is enhanced to support R model. For example, `CREATE_MODEL` and `DROP_MODEL`.
- `MODEL VIEW` to get the R model details about a single model and a partitioned model.
- OML4SQL SQL functions are enhanced to operate with the R model functions. For example, `PREDICTION` and `CLUSTER_ID`.

R model extensibility supports the following OML4SQL functions:

- Association
- Attribute Importance
- Regression
- Classification
- Clustering
- Feature Extraction

28.2 Scoring with R

Learn how to build and score with an Oracle Machine Learning for R model.

For more information, see *Oracle Machine Learning for SQL User's Guide*

28.3 About Algorithm Metadata Registration

Algorithm metadata registration allows for a uniform and consistent approach of registering new algorithm functions and their settings.

Users have the ability to add new R-based algorithms through the registration process. The new algorithms appear as available within Oracle Machine Learning for R and within the appropriate machine learning techniques. Based on the registration metadata, the settings page is dynamically rendered. The advantages are as follows:

- Manage R-based algorithms more easily
- Specify R-based algorithm for model build
- Clean individual properties in JSON structure
- Share R-based algorithm across user

Algorithm metadata registration extends the machine learning model capability of Oracle Machine Learning for SQL.

> **See Also:**
>
> DBMS_DATA_MINING — Algorithm Settings: ALGO_EXTENSIBLE_LANG for a listing and explanation of the available model settings.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- Create Model Using Registration Information
- FETCH_JSON_SCHEMA Procedure
- REGISTER_ALGORITHM Procedure

- JSON Schema for R Extensible Algorithm

28.3.1 Algorithm Metadata Registration

Algorithm metadata registration allows for a uniform and consistent approach of registering new algorithm functions and their settings.

User have the ability to add new algorithms through the `REGISTER_ALGORITHM` procedure registration process. The new algorithms can appear as available within Oracle Machine Learning for SQL for their appropriate machine learning functions. Based on the registration metadata, the settings page is dynamically rendered. Algorithm metadata registration extends the machine learning model capability of OML4SQL.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- FETCH_JSON_SCHEMA Procedure
- REGISTER_ALGORITHM Procedure
- JSON Schema for R Extensible Algorithm

29 Random Forest

Learn how to use Random Forest as a classification algorithm.

- About Random Forest
- Building a Random Forest
- Scoring with Random Forest

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Random Forest
- Automatic Data Preparation
- Model Detail Views for Random Forest
- OML4SQL Examples
- OML4R Random Forest Example
- OML4R GitHub Examples

29.1 About Random Forest

Random Forest is a classification algorithm that builds an **ensemble** (also called **forest**) of trees.

The algorithm builds a number of Decision Tree models and predicts using the ensemble. An individual decision tree is built by choosing a random sample from the training data set as the input. At each node of the tree, only a random sample of predictors is chosen for computing the split point. This introduces variation in the data used by the different trees in the forest. The parameters `RFOR_SAMPLING_RATIO` and `RFOR_MTRY` are used to specify the sample size and number of predictors chosen at each node. Users can use `ODMS_RANDOM_SEED` to set the random seed value before running the algorithm.

Related Topics

- About Decision Tree
  Decision tree is a supervised machine learning algorithm used for classifying data. Decision tree has a tree structure built top-down that has a root node, branches, and leaf nodes.
- Splitting
  The Decision Tree algorithm offers metrics for splitting the cases (records).
- Data Preparation for Decision Tree
  The Decision Tree algorithm manages its own data preparation internally. It does not require pretreatment of the data.

29.2 Building a Random Forest

The Random Forest is built upon existing infrastructure and Application Programming Interfaces (APIs) of Oracle Machine Learning for SQL.

Random forest models provide attribute importance ranking of predictors. The model is built by specifying parameters in the existing APIs. The scoring is performed using the same SQL queries and APIs as the existing classification algorithms. OML4SQL implements a variant of classical Random Forest algorithm. This implementation supports big data sets. The implementation of the algorithm differs in the following ways:

- OML4SQL does not support bagging and instead provides sampling without replacement
- Users have the ability to specify the depth of the tree. Trees are not built to maximum depth.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Random Forest

29.3 Scoring with Random Forest

Learn to score with the Random Forest algorithm.

Scoring with Random Forest is the same as any other classification algorithm. The following functions are supported: `PREDICTION`, `PREDICTION_PROBABILITY`, `PREDICTION_COST`, `PREDICTION_SET`, and `PREDICTION_DETAILS`.

Related Topics

- *Oracle Database SQL Language Reference*

30

Singular Value Decomposition

Learn how to use Singular Value Decomposition, an unsupervised algorithm for feature extraction.

- About Singular Value Decomposition
- Configuring the Algorithm
- Data Preparation for SVD

Related Topics

- Feature Extraction
  Learn how to perform attribute reduction using feature extraction as an unsupervised function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Constants and Settings: Singular Value Decomposition
- Automatic Data Preparation
- Model Detail Views for Singular Value Decompisition
- OML4SQL Examples
- OML4R Singular Value Decomposition Example
- OML4R GitHub Examples

30.1 About Singular Value Decomposition

SVD and the closely-related PCA are well established feature extraction methods that have a wide range of applications. Oracle Machine Learning for SQL implements Singular Value Decomposition (SVD) as a feature extraction algorithm and Principal Component Analysis (PCA) as a special scoring method for SVD models.

SVD and PCA are orthogonal linear transformations that are optimal at capturing the underlying variance of the data. This property is very useful for reducing the dimensionality of high-dimensional data and for supporting meaningful data visualization.

SVD and PCA have a number of important applications in addition to dimensionality reduction. These include matrix inversion, data compression, and the imputation of unknown data values.

30.1.1 Matrix Manipulation

Singular Value Decomposition (SVD) is a factorization method that decomposes a rectangular matrix **X** into the product of three matrices: **U**, **S**, and **V**.

Figure 30-1 Matrix Manipulation

$$\mathbf{X} = \mathbf{USV'}$$

The **U** matrix consists of a set of 'left' orthonormal bases
The **S** matrix is a diagonal matrix
The **V** matrix consists of set of 'right' orthonormal bases

The values in **S** are called singular values. They are non-negative, and their magnitudes indicate the importance of the corresponding bases (components). The singular values reflect the amount of data variance captured by the bases. The first basis (the one with largest singular value) lies in the direction of the greatest data variance. The second basis captures the orthogonal direction with the second greatest variance, and so on.

SVD essentially performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance in the data. This is a useful procedure under the assumption that the observed data has a high signal-to-noise ratio and that a large variance corresponds to interesting data content while a lower variance corresponds to noise.

SVD makes the assumption that the underlying data is Gaussian distributed and can be well described in terms of means and covariances.

30.1.2 Low Rank Decomposition

Singular Value Decomposition (SVD) keeps lower-order bases (the ones with the largest singular values) and ignores higher-order bases (the ones with the smallest singular values) to capture the most important aspects of the data.

To reduce dimensionality, SVD keeps lower-order bases and ignores higher-order bases. The rationale behind this strategy is that the low-order bases retain the characteristics of the data that contribute most to its variance and are likely to capture the most important aspects of the data.

Given a data set **X** (*n*x*m*), where *n* is the number of rows and *m* is the number of attributes, a low-rank SVD uses only *k* components (*k* <= **min**(*m*, *n*)). In typical implementations of SVD, the value of *k* requires a visual inspection of the ranked singular values associated with the individual components. In OML4SQL, SVD automatically estimates the cutoff point, which corresponds to a significant drop in the explained variance.

SVD produces two sets of orthonormal bases (**U** and **V**). Either of these bases can be used as a new coordinate system. In OML4SQL, SVD, **V** is the new coordinate system, and **U** represents the projection of **X** in this coordinate system. The algorithm computes the projection of new data as follows:

Figure 30-2 Computing Projection of New Data

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{V}_k\mathbf{S}_k^{-1}$$

where **X** (*n*x*k*) is the projected data in the reduced data space, defined by the first *k* components, and $\mathbf{V}_k$ and $\mathbf{S}_k$ define the reduced component set.

30.1.3 Scalability

In Oracle Machine Learning for SQL, Singular Value Decomposition (SVD) can process data sets with millions of rows and thousands of attributes. Oracle Machine Learning for SQL automatically recommends an appropriate number of features, based on the data, for dimensionality reduction.

SVD has linear scalability with the number of rows and cubic scalability with the number of attributes when a full decomposition is computed. A low-rank decomposition is typically linear with the number of rows and linear with the number of columns. The scalability with the reduced rank depends on how the rank compares to the number of rows and columns. It can be linear when the rank is significantly smaller or cubic when it is on the same scale.

30.2 Configuring the Algorithm

Several options are available for configuring the Singular Value Decomposition (SVD) algorithm.

Among several options are: settings to control model size and performance, and whether to score with SVD projections or Principal Component Analysis (PCA) projections.

See Also:

DBMS_DATA_MINING — Algorithm Constants and Settings: Singular Value Decomposition for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

30.2.1 Model Size

Learn how a model size is decided based on the rows in the build data and algorithm-specific setting.

The **U** matrix in Singular Value Decomposition has as many rows as the number of rows in the build data. To avoid creating a large model, the **U** matrix persists only when an algorithm-specific setting is enabled. By default the **U** matrix does not persist.

30.2.2 Performance

Singular Value Decomposition can use approximate computations to improve performance.

Approximation may be appropriate for data sets with many columns. An approximate low-rank decomposition provides good solutions at a reasonable computational cost. The quality of the approximation is dependent on the characteristics of the data.

30.2.3 PCA scoring

Learn about configuring Singular Value Decomposition (SVD) to perform Principal Component Analysis (PCA) projections.

SVD models can be configured to perform PCA projections. PCA is closely related to SVD. PCA computes a set of orthonormal bases (principal components) that are ranked by their corresponding explained variance. The main difference between SVD and PCA is that the PCA projection is not scaled by the singular values. The PCA projection to the new coordinate system is given by:

Figure 30-3 PCA Projection Calculation

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{V}_k$$

where

$\tilde{\mathbf{X}}$

(nxk) is the projected data in the reduced data space, defined by the first k components, and $\mathbf{V}_k$ defines the reduced component set.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

30.3 Data Preparation for SVD

Oracle Machine Learning for SQL implements Singular Value Decomposition (SVD) for numerical data and categorical data.

When the build data is scored with SVD, Automatic Data Preparation does nothing. When the build data is scored with Principal Component Analysis (PCA), Automatic Data Preparation shifts the numerical data by mean.

Missing value treatment is not needed, because OML4SQL algorithms handle missing values automatically. SVD replaces numerical missing values with the mean and categorical missing values with the mode. For sparse data (missing values in nested columns), SVD replaces missing values with zeros.

Related Topics

- Prepare the Data

31 Support Vector Machine

Learn how to use Support Vector Machine (SVM), a powerful algorithm based on statistical learning theory.

Oracle Machine Learning for SQL implements SVM for classification, regression, and anomaly detection.

- About Support Vector Machine
- Tuning an SVM Model
- Data Preparation for SVM
- SVM Classification
- One-Class SVM
- SVM Regression

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- Regression
  Learn how to predict a continuous numerical target through regression - the supervised machine learning technique.
- Anomaly Detection
  Learn how to detect rare cases in the data through anomaly detection - an unsupervised function.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: Support Vector Machine
- Automatic Data Preparation
- Model Detail View for Support Vector Machine
- OML4SQL Examples
- OML4R Support Vector Machine Example
- OML4R GitHub Examples
- `Oracle Machine Learning for SQL`

See Also:

Milenova, B.L., Yarmus, J.S., Campos, M.M., "Support Vector Machines in Oracle Database 10*g*: Removing the Barriers to Widespread Adoption of Support Vector Machines", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

31.1 About Support Vector Machine

Support Vector Machine (SVM) is a powerful, state-of-the-art algorithm with strong theoretical foundations based on the Vapnik-Chervonenkis theory.

SVM has strong **regularization** properties. Regularization refers to the generalization of the model to new data.

31.1.1 Advantages of SVM

Support Vector Machine (SVM) implements solvers for scalability and handling large volumes of data.

Oracle Machine Learning for SQL SVM implementation includes two types of solvers, an Interior Point Method (IPM) solver and a Sub-Gradient Descent (SGD) solver. The IPM solver provides stable and accurate solutions, however, it may not be able to handle data of high dimensionality. For high-dimensional and/or large data, for example, text, ratings, and so on, the SGD solver is a better choice. Both solvers have highly scalable parallel implementations and can handle large volumes of data.

31.1.2 Advantages of SVM in Oracle Machine Learning for SQL

Describes advantages of using the Support Vector Machine (SVM) algorithm.

Oracle Machine Learning for SQL has its own proprietary implementation of SVM, which exploits the many benefits of the algorithm while compensating for some of the limitations inherent in the SVM framework. OML4SQL SVM provides the scalability and usability that are needed in a production quality OML4SQL system.

31.1.2.1 Usability

Explains usability for Support Vector Machine (SVM) in Oracle Machine Learning for SQL.

Usability is a major enhancement, because SVM has often been viewed as a tool for experts. The algorithm typically requires data preparation, tuning, and optimization. Oracle Machine Learning minimizes these requirements. You do not need to be an expert to build a quality SVM model in OML4SQL. For example:

- Data preparation is not required in most cases.
- Default tuning parameters are generally adequate.

Related Topics

- Data Preparation for SVM
  Support Vector Machine (SVM) uses normalization and missing value treatment for data preparation.
- Tuning an SVM Model
  The Support Vector Machine (SVM) algorithm has built-in mechanisms that automatically choose appropriate settings based on the data.

31.1.2.2 Scalability

Learn how to scale the data for Support Vector Machine (SVM).

When dealing with very large data sets, sampling is often required. However, sampling is not required with Oracle Machine Learning for SQL SVM, because the algorithm itself uses stratified sampling to reduce the size of the training data as needed.

OML4SQL SVM is highly optimized. It builds a model incrementally by optimizing small working sets toward a global solution. The model is trained until convergence on the current working set, then the model adapts to the new data. The process continues iteratively until the convergence conditions are met. The Gaussian kernel uses caching techniques to manage the working sets.

Related Topics

- Kernel-Based Learning
  Learn about kernal-based functions to transform the input data for Support Vector Machine (SVM).

31.1.3 Kernel-Based Learning

Learn about kernal-based functions to transform the input data for Support Vector Machine (SVM).

SVM is a kernel-based algorithm. A **kernel** is a function that transforms the input data to a high-dimensional space where the problem is solved. Kernel functions can be linear or nonlinear.

Oracle Machine Learning for SQL supports linear and Gaussian (nonlinear) kernels.

In OML4SQL, the **linear kernel** function reduces to a linear equation on the original attributes in the training data. A linear kernel works well when there are many attributes in the training data.

The **Gaussian kernel** transforms each case in the training data to a point in an *n*-dimensional space, where *n* is the number of cases. The algorithm attempts to separate the points into subsets with homogeneous target values. The Gaussian kernel uses nonlinear separators, but within the kernel space it constructs a linear equation.

> **Note:**
>
> Active Learning is not relevant in Oracle Database 12*c* Release 2 and later. A setting similar to Active Learning is `ODMS_SAMPLING`.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.2 Tuning an SVM Model

The Support Vector Machine (SVM) algorithm has built-in mechanisms that automatically choose appropriate settings based on the data.

You may need to override the system-determined settings for some domains.

Settings pertain to regression, classification, and anomaly detection unless otherwise specified.

See Also:

DBMS_DATA_MINING —Algorithm Settings: Support Vector Machine for a listing and explanation of the available model settings.

Note:

The term hyperparameter is also interchangeably used for model setting.

31.3 Data Preparation for SVM

Support Vector Machine (SVM) uses normalization and missing value treatment for data preparation.

The SVM algorithm operates natively on numeric attributes. SVM uses z-score normalization on numeric attributes. The normalization occurs only for two-dimensional numeric columns (not nested). The algorithm automatically "explodes" categorical data into a set of binary attributes, typically one per category value. For example, a character column for marital status with values `married` or `single` is transformed to two numeric attributes: `married` and `single`. The new attributes can have the value `1` (true) or `0` (false).

When there are missing values in columns with simple data types (not nested), SVM interprets them as missing at random. The algorithm automatically replaces missing categorical values with the mode and missing numerical values with the mean.

When there are missing values in the nested columns, SVM interprets them as sparse. The algorithm automatically replaces sparse numerical data with zeros and sparse categorical data with zero vectors.

31.3.1 Normalization

Transform data through normalization for Support Vector Machine (SVM).

SVM require the normalization of numeric input. Normalization places the values of numeric attributes on the same scale and prevents attributes with a large original scale from biasing the solution. Normalization also minimizes the likelihood of overflows and underflows.

31.3.2 SVM and Automatic Data Preparation

You can prepare data by treating and transforming data manually or through Automatic Data Preparation (ADP) for Support Vector Machine (SVM).

The SVM algorithm automatically handles missing value treatment and the transformation of categorical data, but normalization and outlier detection must be handled by ADP or prepared manually. ADP performs min-max normalization for SVM.

> **Note:**
>
> Oracle recommends that you use ADP with SVM. The transformations performed by ADP are appropriate for most models.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

31.4 SVM Classification

Support Vector Machine (SVM) classification is based on the concept of decision planes that define decision boundaries.

A decision plane is one that separates between a set of objects having different class memberships. SVM finds the vectors ("support vectors") that define the separators giving the widest separation of classes.

SVM classification supports both binary, multiclass, and multitarget classification. Multitarget alllows multiple class labels to be associated with a single row. The target type is a collection of type `ORA_MINING_VARCHAR2_NT`.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.4.1 Class Weights

Learn when to implement class weights to a data in Support Vector Machine (SVM).

In SVM classification, weights are a biasing mechanism for specifying the relative importance of target values (classes).

SVM models are automatically initialized to achieve the best average prediction across all classes. However, if the training data does not represent a realistic distribution, you can bias the model to compensate for class values that are under-represented. If you increase the weight for a class, then the percent of correct predictions for that class must increase.

Related Topics

- Priors and Class Weights
  Learn about Priors and Class Weights in a classification model to produce a useful result.

31.5 One-Class SVM

Support Vector Machine (SVM) as a one-class classifier is used for detecting anomalies.

Oracle Machine Learning for SQL uses SVM as the one-class classifier for anomaly detection. When SVM is used for anomaly detection, it has the classification machine learning technique but no target.

One-class SVM models, when applied, produce a prediction and a probability for each case in the scoring data. If the prediction is 1, the case is considered typical. If the prediction is 0, the case is considered anomalous. This behavior reflects the fact that the model is trained with normal data.

You can specify the percentage of the data that you expect to be anomalous with the `SVMS_OUTLIER_RATE` build setting. If you have some knowledge that the number of "suspicious" cases is a certain percentage of your population, then you can set the outlier rate to that percentage. The model approximately identifies that many "rare" cases when applied to the general population.

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING - Algorithm Settings: Support Vector Machine
- Automatic Data Preparation
- Model Detail View for Support Vector Machine
- OML4SQL Examples
- OML4R SVM Example
- OML4R GitHub Examples

31.6 SVM Regression

Learn how to use epsilon-insensitivity loss function to solve regression problems in Support Vector Machine (SVM).

SVM uses an epsilon-insensitive loss function to solve regression problems.

SVM regression tries to find a continuous function such that the maximum number of data points lie within the epsilon-wide insensitivity tube. Predictions falling within epsilon distance of the true target value are not interpreted as errors.

The epsilon factor is a regularization setting for SVM regression. It balances the margin of error with model robustness to achieve the best generalization to new data.

Related Topics

- SVM Model Settings

32
XGBoost

XGBoost is highly-efficient, scalable machine learning algorithm for regression and classification that makes available the XGBoost Gradient Boosting open source package.

- About XGBoost
- XGBoost Feature Constraints
- XGBoost AFT Model
- Ranking Methods
- Scoring with XGBoost

Related Topics

- Classification
  Learn how to predict a categorical target through classification - the supervised machine learning technique.
- Regression
  Learn how to predict a continuous numerical target through regression - the supervised machine learning technique.
- Ranking
  Ranking is a regression machine learning technique.
- DBMS_DATA_MINING - Model Settings
- DBMS_DATA_MINING — Algorithm Settings: XGBoost
- Automatic Data Preparation
- Model Detail Views for XGBoost
- OML4SQL Examples

32.1 About XGBoost

Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.

Oracle Machine Learning for SQL XGBoost is a scalable gradient tree boosting system that supports both classification and regression. It makes available the open source gradient boosting framework.

You can use XGBoost as a stand-alone predictor or incorporate it into real-world production pipelines for a wide range of problems such as ad click-through rate prediction, hazard risk prediction, web text classification, and so on.

The Oracle Machine Learning for SQL XGBoost algorithm takes three types of parameters: general parameters, booster parameters, and task parameters. You set the parameters through the model settings table. The algorithm supports most of the settings of the open source project.

Through XGBoost, Oracle Machine Learning for SQL supports a number of different classification and regression specifications, ranking models, and survival models. Binary and multiclass models are supported under the classification machine learning technique while regression, ranking, count, and survival are supported under the regression machine learning technique.

XGBoost also supports partitioned models and internalizes the data preparation. Currently, XGBoost is available only on Oracle Database Linux platform.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: XGBoost
- XGBoost: A Scalable Tree Boosting System, by Tianqi Chen and Carlos Guestrin
- XGBoost on GitHub

32.2 XGBoost Feature Constraints

Feature interaction constraints allow users to specify which variables can and cannot interact. By focusing on key interactions and eliminating noise, it aids in improving predicting performance. This, in turn, may lead to more generalized predictions.

The feature interaction constraints are described in terms of groupings of features that are allowed to interact. Variables that appear together in a traversal path in decision trees interact with one another because the condition of a child node is dependent on the condition of the parent node. These additional controls on model fit are beneficial to users who have a good understanding of the modeling task, including domain knowledge. Oracle Machine Learning for SQL supports more of the available XGBoost capabilities once these constraints are applied.

Monotonic constraints allow you to impose monotonicity constraints on the features in your boosted model. There may be a strong prior assumption that the genuine relationship is constrained in some way in many circumstances. This could be owing to commercial factors (just specific feature interactions are of interest) or the type of scientific subject under investigation. A typical form of constraint is that some features have a monotonic connection to the predicted response. In these situations, monotonic constraints may be employed to improve the model's prediction performance. For example, let *X* be the feature vector with features *[x1,…, xi , …, xn]* and *ƒ(X)* be the prediction response. Then `ƒ(X) ≤ ƒ(X')` whenever `xi ≤ xi'` is an increasing constraint; `ƒ(X) ≥ ƒ(X')` whenever `xi ≤ xi'` is a decreasing constraint. These feature constraints are listed in DBMS_DATA_MINING — Algorithm Settings: XGBoost.

The following example displays the code snippet for defining feature constraints using the XGBoost algorithm. XGBoost `interaction_constraints` setting is used to specify the interaction constraints. The example predicts customers most likely to respond positively for an affinity card loyalty program.

```
-----------------------------------------------------------------------
--   Build a Classification Model using Interaction Contraints
-----------------------------------------------------------------------
-- The interaction constraints setting can be used to specify
permitted
-- interactions in the model. The constraints must be specified
-- in the form of nested list, where each inner list is a group of
-- features (column names) that are allowed to interact with each
```

```
other.
-- For example, assume x0, x1, x2, x3, x4, x5 and x6 are
-- the feature names (column names) of interest.
-- Then setting value [[x0,x1,x2],[x0,x4],[x5,x6]] specifies that:
--    * Features x0, x1 and x2 are allowed to interact with each other
--    but with no other feature.
--    * Features x0 & x4 are allowed to interact with one another
--    but with no other feature.
--    * Features x5 and x6 are allowed to interact with each other
--    but with no other feature.
-------------------------------------------------------------------------

BEGIN DBMS_DATA_MINING.DROP_MODEL('XGB_CLASS_MODEL_INTERACTIONS');
EXCEPTION WHEN OTHERS THEN NULL; END;
/

DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME')    := 'ALGO_XGBOOST';
    v_setlst('PREP_AUTO')    := 'ON';
    v_setlst('max_depth')    := '2';
    v_setlst('eta')          := '1';
    v_setlst('num_round')    := '100';
    v_setlst('interaction_constraints')  := '[[YRS_RESIDENCE, OCCUPATION],
                                              [OCCUPATION, Y_BOX_GAMES],
                                              [BULK_PACK_DISKETTES,
                                             BOOKKEEPING_APPLICATION]]';

    DBMS_DATA_MINING.CREATE_MODEL2(
            MODEL_NAME          => 'XGB_CLASS_MODEL_INTERACTIONS',
            MINING_FUNCTION     => 'CLASSIFICATION',
            DATA_QUERY          => 'SELECT * FROM TRAIN_DATA_CLAS',
            SET_LIST            =>  v_setlst,
            CASE_ID_COLUMN_NAME => 'CUST_ID',
            TARGET_COLUMN_NAME  => 'AFFINITY_CARD');

    DBMS_OUTPUT.PUT_LINE('Created model: XGB_CLASS_MODEL_INTERACTIONS');
END;
/
```

To view the complete example, see https://github.com/oracle-samples/oracle-db-examples/tree/main/machine-learning/sql/23c.

32.3 XGBoost AFT Model

Survival analysis is a field of statistics that examines the time elapsed between one or more occurrences, such as death in biological organisms and failure in mechanical systems.

The goals of survival analysis include evaluating patterns of event times, comparing distributions of survival times in different groups of people, and determining if and how much certain factors affect the likelihood of an event of interest. The existence of censored data is an important feature of survival analysis. If a person does not experience an event within the observation period, they are labeled as censored. **Censoring** is a type of missing data problem in which the time to event is not recorded for a variety of reasons, such as the study being terminated before all enrolled subjects have demonstrated the event of interest, or the subject leaving the study before experiencing an event. Right censoring is defined as knowing only the lower limit *l* for the genuine event time *T* such that $T > l$. Right censoring will take place, for example, for those subjects whose birth date is known but who are still living when they are lost to follow-up or when the study concludes. We frequently come upon data that has been right-censored. The data is said to be left-censored if the event of interest occurred before the subject was included in the study but the exact date is unknown. Interval censoring occurs when an occurrence can only be described as occurring between two observations or examinations.

The Cox proportional hazards model and the Accelerated Failure Time (AFT) model are two major survival analysis methods. Oracle Machine Learning for SQL supports both these models.

Cox regression works for right censored survival time data. The hazard rate is the risk of failure (that is, the risk or likelihood of suffering the event of interest) in a Cox proportional hazards regression model, assuming that the subject has lived up to a particular time. The Cox predictions are returned on a hazard ratio scale. A Cox proportional hazards model has the following form:

$h\ (t,x) = h_0(t)e^{\beta x}$

Where $h(t)$ is the baseline hazard, x is a covariate, and β is an estimated parameter that represents the covariate's effect on the outcome. A Cox proportional hazards model's estimated amount is understood as relative risk rather than absolute risk.

The AFT model fits models to data that can be censored to the left, right, or interval. The AFT model, which models time to an event of interest, is one of the most often used models in survival analysis. AFT is a parametric (it assumes the distribution of response data) survival model. The outcome of AFT models has a physical interpretation that is intuitive. The model has the following form:

$ln\ Y = < W, X> + \sigma Z$

Where X is the vector in R^d representing the features. W is a vector consisting of d coefficients, each corresponding to a feature. $<W, X>$ is the usual dot product in R^d. Y is the random variable modeling the output label. Z is a random variable of a known probability distribution. Common choices are the normal distribution, the logistic distribution, and the extreme distribution. It represents the “noise”. σ is a parameter that scales the size of noise.

AFT model that works with XGBoost or gradient boosting has the following form:

$ln\ Y = T(x) + \sigma Z$

Where $T(x)$ represents the output of a decision tree ensemble, using the supplied input x. Since Z is a random variable, you have a likelihood defined for the expression $lnY=T(x)+\sigma Z$. As a result, XGBoost's purpose is to maximize (log) likelihood by fitting a suitable tree ensemble $T(x)$.

The AFT parameters are listed in DBMS_DATA_MINING — Algorithm Settings: XGBoost.

The following example displays code snippet of survival analysis using the XGBoost algorithm. In this example, a `SURVIVAL_DATA` table is created that contains data for survival analysis. XGBoost AFT settings `aft_right_bound_column_name`, aft_loss_distribution, and aft_loss_distribution_scale are illustrated in this example.

```
-----------------------------------------------------------------------------
--          Create a data table with left and right bound columns
-----------------------------------------------------------------------------

-- The data table 'SURVIVAL_DATA' contains both exact data point and
-- right-censored data point. The left bound column is set by
-- parameter target_column_name. The right bound column is set
-- by setting aft_right_bound_column_name.

-- For right censored data point, the right bound is infinity,
-- which is represented as NULL in the right bound column.

BEGIN EXECUTE IMMEDIATE 'DROP TABLE SURVIVAL_DATA';
EXCEPTION WHEN OTHERS THEN NULL; END;
/
CREATE TABLE SURVIVAL_DATA (INST NUMBER, LBOUND NUMBER, AGE NUMBER,
                            SEX NUMBER, PHECOG NUMBER, PHKARNO NUMBER,
                            PATKARNO NUMBER, MEALCAL NUMBER, WTLOSS NUMBER,
                            RBOUND NUMBER);
INSERT INTO SURVIVAL_DATA VALUES(26, 235, 63, 2, 0, 100,  90,  413,  0,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(22, 444, 75, 2, 2,  70,  70,  438,  8,
444);
INSERT INTO SURVIVAL_DATA VALUES(16, 806, 44, 1, 1,  80,  80, 1025,  1,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(16, 551, 77, 2, 2,  80,  60,  750, 28,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(3,  202, 50, 2, 0, 100, 100,  635,  1,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(7,  583, 68, 1, 1,  60,  70, 1025,  7,
583);
INSERT INTO SURVIVAL_DATA VALUES(32, 135, 60, 1, 1,  90,  70, 1275,  0,
135);
INSERT INTO SURVIVAL_DATA VALUES(21, 237, 69, 1, 1,  80,  70, NULL, NULL,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(26, 356, 53, 2, 1,  90,  90, NULL,   2,
NULL);
INSERT INTO SURVIVAL_DATA VALUES(13, 387, 56, 1, 2,  80,  60, 1075, NULL,
387);

-----------------------------------------------------------------------------
--              Build an XGBoost survival model with survival:aft
-----------------------------------------------------------------------------

BEGIN DBMS_DATA_MINING.DROP_MODEL('XGB_SURVIVAL_MODEL');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME')                    := 'ALGO_XGBOOST';
    v_setlst('max_depth')                    := '6';
    v_setlst('eval_metric')                  := 'aft-nloglik';
```

```
    v_setlst('num_round')                    := '100';
    v_setlst('objective')                    := 'survival:aft';
    v_setlst('aft_right_bound_column_name')  := 'rbound';
    v_setlst('aft_loss_distribution')        := 'normal';
    v_setlst('aft_loss_distribution_scale')  := '1.20';
    v_setlst('eta')                          := '0.05';
    v_setlst('lambda')                       := '0.01';
    v_setlst('alpha')                        := '0.02';
    v_setlst('tree_method')                  := 'hist';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'XGB_SURVIVAL_MODEL',
        MINING_FUNCTION     => 'REGRESSION',
        DATA_QUERY          => 'SELECT * FROM SURVIVAL_DATA',
        TARGET_COLUMN_NAME  => 'LBOUND',
        CASE_ID_COLUMN_NAME =>  NULL,
        SET_LIST            =>  v_setlst);
END;
/
```

To view the complete example, see https://github.com/oracle-samples/oracle-db-examples/blob/main/machine-learning/sql/23c/oml4sql-survival-analysis-xgboost.sql.

32.4 Ranking Methods

Oracle Machine Learningsupports pairwise and listwise ranking methods through XGBoost.

For a training data set, in a number of sets, each set consists of objects and labels representing their ranking. A ranking function is constructed by minimizing a certain loss function on the training data. Using test data, the ranking function is applied to get a ranked list of objects. Ranking is enabled for XGBoost using the regression function. OML4SQL supports pairwise and listwise ranking methods through XGBoost.

Pairwise ranking: This approach regards a pair of objects as the learning instance. The pairs and lists are defined by supplying the same `case_id` value. Given a pair of objects, this approach gives an optimal ordering for that pair. Pairwise losses are defined by the order of the two objects. In OML4SQL, the algorithm uses LambdaMART to perform pairwise ranking with the goal of minimizing the average number of inversions in ranking.

Listwise ranking: This approach takes multiple lists of ranked objects as learning instance. The items in a list must have the same `case_id`. The algorithm uses LambdaMART to perform list-wise ranking.

See Also:

- "Ranking Measures and Loss Functions in Learning to Rank" a research paper presentation at https://www.researchgate.net/
- *Oracle Database PL/SQL Packages and Types Reference* for a listing and explanation of the available model settings for XGBoost.

> **Note:**
>
> The term hyperparameter is also interchangeably used for model setting.

Related Topics

- About XGBoost
  Oracle Machine Learning for SQL XGBoost prepares training data, invokes XGBoost, builds and persists a model, and applies the model for prediction.
- DBMS_DATA_MINING — Algorithm Settings: XGBoost

32.5 Scoring with XGBoost

Learn how to score with XGBoost.

The SQL scoring functions supported for a classification XGBoost model are `PREDICTION`, `PREDICTION_COST`, `PREDICTION_DETAILS`, `PREDICTION_PROBABILITY`, and `PREDICTION_SET`.

The scoring functions supported for a regression XGBoost model are `PREDICTION` and `PREDICTION_DETAILS`.

The prediction functions return the following information:

- `PREDICTION` returns the predicted value.
- `PREDICTION_COST` returns a measure of cost for a given prediction as an Oracle NUMBER. (classification only)
- `PREDICTION_DETAILS` returns the SHAP (SHapley Additive exPlanation) contributions.
- `PREDICTION_PROBABILITY` returns the probability for a given prediction. (classification only)
- `PREDICTION_SET` returns the prediction and the corresponding prediction probability for each observation. (classification only)

Related Topics

- SQL Scoring Functions

Part IV

Using the Oracle Machine Learning for SQL API

Learn how to use Oracle Machine Learning for SQL application programming interface.

- Oracle Machine Learning With SQL
- About the Oracle Machine Learning for SQL API
- Prepare the Data
- Create a Model
- Scoring and Deployment
- Machine Learning Operations on Unstructured Text
- Administrative Tasks for Oracle Machine Learning for SQL
- Examples

33

Oracle Machine Learning With SQL

Learn how to solve business problems using the Oracle Machine Learning for SQL application programming interface (API).

33.1 Highlights of the Oracle Machine Learning for SQL API

Learn about the advantages of OML4SQL application programming interface (API).

Machine learning is a valuable technology in many application domains. It has become increasingly indispensable in the private sector as a tool for optimizing operations and maintaining a competitive edge. Machine learning also has critical applications in the public sector and in scientific research. However, the complexities of machine learning application development and the complexities inherent in managing and securing large stores of data can limit the adoption of machine learning technology.

OML4SQL is uniquely suited to addressing these challenges. The machine learning engine is implemented in the database kernel, and the robust administrative features of Oracle Database are available for managing and securing the data. While supporting a full range of machine learning algorithms and procedures, the API also has features that simplify the development of machine learning applications.

The OML4SQL API consists of extensions to Oracle SQL, the native language of the database. The API offers the following advantages:

- Scoring in the context of SQL queries. Scoring can be performed dynamically or by applying machine learning models.
- Automatic Data Preparation (ADP) and embedded transformations.
- Model transparency. Algorithm-specific queries return details about the attributes that were used to create the model.
- Scoring transparency. Details about the prediction, clustering, or feature extraction operation can be returned with the score.
- Simple routines for predictive analytics.
- A workflow-based graphical user interface (GUI) within Oracle SQL Developer. You can download SQL Developer free of charge from the following site:

  `Oracle Data Miner`

> **Note:**
>
> The examples in this publication are taken from the OML4SQL examples that are available on GitHub. For information on the examples, see About the OML4SQL Examples.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

33.2 Example: Predicting Likely Candidates for a Sales Promotion

This example shows `PREDICTION` query to target customers in Brazil for a special promotion that offers coupons and an affinity card.

The query uses data on marital status, education, and income to predict the customers who are most likely to take advantage of the incentives. The query applies a Decision Tree model called dt_sh_clas_sample to score the customer data. The model is created by the `oml4sql-classification-decision-tree.sql` example.

Example 33-1 Predict Best Candidates for an Affinity Card

```
SELECT cust_id
  FROM mining_data_apply_v
  WHERE
      PREDICTION(dt_sh_clas_sample
                   USING cust_marital_status, education, cust_income_level ) = 1
  AND country_name IN 'Brazil';
```

The output is as follows:

```
CUST_ID
----------
    100404
    100607
    101113
```

The same query, but with a bias to favor false positives over false negatives, is shown here.

```
SELECT cust_id
  FROM mining_data_apply_v
  WHERE
      PREDICTION(dt_sh_clas_sample COST MODEL
                   USING cust_marital_status, education, cust_income_level ) = 1
  AND country_name IN 'Brazil';
```

The output is as follows:

```
CUST_ID
----------
    100139
    100163
    100275
    100404
    100607
    101113
    101170
    101463
```

The `COST MODEL` keywords cause the cost matrix associated with the model to be used in making the prediction. The cost matrix, stored in a table called `dt_sh_sample_costs`, specifies that a false negative is eight times more costly than a false positive. Overlooking a likely candidate for the promotion is far more costly than including an unlikely candidate.

```
SELECT * FROM dt_sh_sample_cost;
```

The output is as follows:

```
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1          1
                  1                      0          8
                  1                      1          0
```

33.3 Example: Analyzing Preferred Customers

The examples in this section reveal information about customers who use affinity cards or are likely to use affinity cards.

Example 33-2   Find Demographic Information About Preferred Customers

This query returns the gender, age, and length of residence of typical affinity card holders. The anomaly detection model, SVMO_SH_Clas_sample, returns `1` for typical cases and `0` for anomalies. The demographics are predicted for typical customers only; outliers are not included in the sample. The model is created by the `oml4sql-anomaly-detection-1class-svm.sql` example.

```
SELECT cust_gender, round(avg(age)) age,
       round(avg(yrs_residence)) yrs_residence,
       count(*) cnt
FROM mining_data_one_class_v
WHERE PREDICTION(SVMO_SH_Clas_sample using *) = 1
GROUP BY cust_gender
ORDER BY cust_gender;
```

The output is as follows:

```
CUST_GENDER          AGE YRS_RESIDENCE        CNT
------------ ---------- ------------- ----------
F                     40             4         36
M                     45             5        304
```

Example 33-3 Dynamically Identify Customers Who Resemble Preferred Customers

This query identifies customers who do not currently have an affinity card, but who share many of the characteristics of affinity card holders. The `PREDICTION` and `PREDICTION_PROBABILITY` functions use an `OVER` clause instead of a predefined model to classify the customers. The predictions and probabilities are computed dynamically.

```
SELECT cust_id, pred_prob
 FROM
  (SELECT cust_id, affinity_card,
    PREDICTION(FOR TO_CHAR(affinity_card) USING *) OVER () pred_card,
    PREDICTION_PROBABILITY(FOR TO_CHAR(affinity_card),1 USING *) OVER ()
pred_prob
   FROM mining_data_build_v)
 WHERE affinity_card = 0
  AND pred_card = 1
 ORDER BY pred_prob DESC;
```

The output is as follows:

```
  CUST_ID PRED_PROB
---------- ---------
    102434       .96
    102365       .96
    102330       .96
    101733       .95
    102615       .94
    102686       .94
    102749       .93
.
.
.
.
    102580       .52
    102269       .52
    102533       .51
    101604       .51
    101656       .51

226 rows selected.
```

Example 33-4 Predict the Likelihood that a New Customer Becomes a Preferred Customer

This query computes the probability of a first-time customer becoming a preferred customer (an affinity card holder). This query can be run in real time at the point of sale.

The new customer is a 44-year-old American executive who has a bachelors degree and earns more than $300,000/year. He is married, lives in a household of 3, and has lived in the same residence for the past 6 years. The probability of this customer becoming a typical affinity card holder is only 5.8%.

```
SELECT PREDICTION_PROBABILITY(SVMO_SH_Clas_sample, 1 USING
                             44 AS age,
                             6 AS yrs_residence,
                             'Bach.' AS education,
                             'Married' AS cust_marital_status,
                             'Exec.' AS occupation,
                             'United States of America' AS country_name,
                             'M' AS cust_gender,
                             'L: 300,000 and above' AS cust_income_level,
                             '3' AS houshold_size
                             ) prob_typical
FROM DUAL;
```

The output is as follows:

```
PROB_TYPICAL
------------
  5.8
```

Example 33-5 Use Predictive Analytics to Find Top Predictors

The DBMS_PREDICTIVE_ANALYTICS PL/SQL package contains routines that perform simple machine learning operations without a predefined model. In this example, the EXPLAIN routine computes the top predictors for affinity card ownership. The procedure does not create a model that can be stored in the database for further exploration. Automatic Data Preparation is also performed behind the scenes. The results show that household size, marital status, and age are the top three predictors.

```
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'mining_data_test_v',
        explain_column_name  => 'affinity_card',
        result_table_name    => 'cust_explain_result');
END;
/

SELECT * FROM cust_explain_result
  WHERE rank < 4;
```

The output is as follows:

```
ATTRIBUTE_NAME             ATTRIBUTE_SUBNAME      EXPLANATORY_VALUE        RANK
-------------------------- ---------------------- ------------------ ----------
```

```
HOUSEHOLD_SIZE                                                    .209628541
  1
CUST_MARITAL_STATUS                                               .199794636
  2
AGE                                                               .111683067
  3
```

Another way to arrive at top predictors for affinity ownership is by using attribute importance mining function. Create a model with the Minimum Description Length algorithm. Define `mining_function` as `ATTRIBUTE_IMPORTANCE`. You can then query the `DM$VA` model detail view to get the top three predictors.

```
BEGIN DBMS_DATA_MINING.DROP_MODEL('AI_EXPLAIN_OUTPUT');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME') := 'ALGO_AI_MDL';
    V_setlst('PREP_AUTO') := 'ON';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME => 'AI_EXPLAIN_OUTPUT',
        MINING_FUNCTION => 'ATTRIBUTE_IMPORTANCE',
        DATA_QUERY => 'select * from mining_data_test_v',
        SET_LIST => v_setlst,
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        TARGET_COLUMN_NAME => 'AFFINITY_CARD');
END;


Find the top 3 predictors from the DM$VA model detail view:
SELECT ATTRIBUTE_NAME, ATTRIBUTE_IMPORTANCE_VALUE, ATTRIBUTE_RANK FROM
DM$VAAI_EXPLAIN_OUTPUT;
```

The output is as follows:

```
ATTRIBUTE_NAME            ATTRIBUTE_IMPORTANCE_VALUE
ATTRIBUTE_RANK
HOUSEHOLD_SIZE            0.16154338717879052
1
CUST_MARITAL_STATUS       0.1561477632217005
2
AGE                       0.08440594628406521
3
```

33.4 Example: Segmenting Customer Data

The examples in this section use an Expectation Maximization clustering model to segment the customer data based on common characteristics.

Example 33-6 Compute Customer Segments

This query computes natural groupings of customers and returns the number of customers in each group. The em_sh_clus_sample model is created by the `oml4sql-clustering-expectation-maximization.sql` example.

```
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
  FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;
```

The output is as follows:

```
      CLUS        CNT
---------- ----------
         9        311
         3        294
         7        215
        12        201
        17        123
        16        114
        14         86
        19         64
        15         56
        18         36
```

Example 33-7 Find the Customers Who Are Most Likely To Be in the Largest Segment

The query in Example 33-6 shows that segment 9 has the most members. The following query lists the five customers who are most likely to be in segment 9.

```
SELECT cust_id
FROM (SELECT cust_id, RANK() over (ORDER BY prob DESC, cust_id) rnk_clus2
  FROM (SELECT cust_id,
          ROUND(CLUSTER_PROBABILITY(em_sh_clus_sample, 9 USING *),3) prob
          FROM mining_data_apply_v))
WHERE rnk_clus2 <= 5
ORDER BY rnk_clus2;
```

The output is as follows:

```
   CUST_ID
----------
    100002
    100012
    100016
    100019
    100021
```

Example 33-8 Find Key Characteristics of the Most Representative Customer in the Largest Cluster

The query in Example 33-7 lists customer 100002 first in the list of likely customers for segment 9. The following query returns the five characteristics that are most significant in determining the assignment of customer 100002 to segments with probability > 20% (only segment 9 for this customer).

```
SELECT S.cluster_id, probability prob,
       CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 using T.*) det
 FROM
  (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
    FROM mining_data_apply_v v
    WHERE cust_id = 100002) T,
 TABLE(T.pset) S
 ORDER BY 2 desc;
```

The output is as follows:

```
CLUSTER_ID    PROB DET
---------- -------
-----------------------------------------------------------------------
---------
         9  1.0000 <Details algorithm="Expectation Maximization"
cluster="9">
                   <Attribute name="YRS_RESIDENCE" actualValue="4"
weight="1" rank="1"/>
                   <Attribute name="EDUCATION" actualValue="Bach."
weight="0" rank="2"/>
                   <Attribute name="AFFINITY_CARD" actualValue="0"
weight="0" rank="3"/>
                   <Attribute name="BOOKKEEPING_APPLICATION"
actualValue="1" weight="0" rank="4"/>
                   <Attribute name="Y_BOX_GAMES" actualValue="0"
weight="0" rank="5"/>
                   </Details>
```

33.5 Example : Comparison of Texts Using an ESA Model

The examples shows the `FEATURE_COMPARE` function comparing texts for semantic relatedness (similarity) using the Explicit Semantic Analysis (ESA) prebuilt Wikipedia-based model, which extracts topics and compares text.

The examples shows an ESA model built against a prebuilt Wiki data set rendering over 200,000 features. The documents are analyzed as text and the document titles are given as the feature IDs. In the first example, the pair of sentence scores higher because Nick Price is a golfer born in South Africa.

Similar Texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA
tour golfers from South Africa' text AND USING 'Nick Price won the
2002 Mastercard Colonial Open' text) similarity FROM DUAL;
```

The output is as follows:

```
SIMILARITY
```

```
----------
      .110
```

The output metric shows distance calculation. Therefore, smaller number represent more similar texts. So, `1` minus the distance in the queries result in similarity.

Dissimilar Texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA tour
golfers from South Africa' text AND USING 'John Elway played quarterback for
the Denver Broncos' text) similarity FROM DUAL;
```

The output is as follows:

```
SIMILARITY
----------
      .004
```

34 About the Oracle Machine Learning for SQL API

Overview of the OML4SQL application programming interface (API) components.

- About Oracle Machine Learning Models
- Oracle Machine Learning Data Dictionary Views
- Oracle Machine Learning Modeling, Transformations, and Convenience Functions
- Oracle Machine Learning for SQL Scoring Functions
- Oracle Machine Learning for SQL Statistical Functions

34.1 About Oracle Machine Learning Models

Machine learning models are database schema objects that perform machine learning techniques.

As with all schema objects, access to machine learning models is controlled by database privileges. Models can be exported and imported. They support comments and they can be tracked in the Oracle Database auditing system.

Machine learning models are created by the `CREATE_MODEL2` or the `CREATE_MODEL` procedures in the `DBMS_DATA_MINING` PL/SQL package. Models are created for a specific machine learning technique, and they use a specific algorithm to perform that function. **Machine learning technique** is a term that refers to a class of machine learning problems to be solved. Examples of machine learning techniques are: regression, classification, attribute importance, clustering, anomaly detection, and feature selection. OML4SQL supports one or more algorithms for each machine learning technique.

Along with the machine learning technique, in the `CREATE_MODEL2` procedure, you can specify an algorithm and other characteristics of a model. In `CREATE_MODEL` procedure you can specify a settings table to specify an algorithm and other characteristics of a model. Some settings are general, some are specific to a machine learning technique, and some are specific to an algorithm.

> **Note:**
>
> Most types of machine learning models can be used to score data. However, it is possible to score data without applying a model. Dynamic scoring and predictive analytics return scoring results without a user-supplied model. They create and apply transient models that are not visible to you.

34.2 Oracle Machine Learning Data Dictionary Views

Lists Oracle Machine Learning data dictionary views.

The data dictionary views for Oracle Machine Learning are listed in the following table. A database administrator (DBA) and USER versions of the views are also available.

Table 34-1 Data Dictionary Views for Oracle Machine Learning

View Name	Description
ALL_MINING_MODELS	Provides information about all accessible machine learning models
ALL_MINING_MODEL_ATTRIBUTES	Provides information about the attributes of all accessible machine learning models
ALL_MINING_MODEL_PARTITIONS	Provides information about the partitions of all accessible partitioned machine learning models
ALL_MINING_MODEL_SETTINGS	Provides information about the configuration settings for all accessible machine learning models
ALL_MINING_MODEL_VIEWS	Provides information about the model views for all accessible machine learning models
ALL_MINING_MODEL_XFORMS	Provides the user-specified transformations embedded in all accessible machine learning models.

34.2.1 ALL_MINING_MODELS

Describes an example of `ALL_MINING_MODELS` and shows a sample query.

The following example describes `ALL_MINING_MODELS` and shows a sample query.

Example 34-1 ALL_MINING_MODELS

```
 describe ALL_MINING_MODELS
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 MINING_FUNCTION                                    VARCHAR2(30)
 ALGORITHM                                          VARCHAR2(30)
 CREATION_DATE                             NOT NULL DATE
 BUILD_DURATION                                     NUMBER
 MODEL_SIZE                                         NUMBER
 BUILD_SOURCE                        CLOB
 PARTITIONED                                        VARCHAR2(3)
 COMMENTS                                           VARCHAR2(4000)
```

The following query returns the models accessible to you that use the Support Vector Machine algorithm.

```
SELECT mining_function, model_name
    FROM all_mining_models
    WHERE algorithm = 'SUPPORT_VECTOR_MACHINES'
    ORDER BY mining_function, model_name;
```

```
MINING_FUNCTION
MODEL_NAME
-------------------------
--------------------
CLASSIFICATION
PART2_CLAS_SAMPLE
CLASSIFICATION
PART_CLAS_SAMPLE
CLASSIFICATION
SVMC_SH_CLAS_SAMPLE
CLASSIFICATION
SVMO_SH_CLAS_SAMPLE
CLASSIFICATION
T_SVM_CLAS_SAMPLE
REGRESSION                SVMR_SH_REGR_SAMPLE
```

The models are created by the following examples:

- PART2_CLAS_SAMPLE by `oml4sql-partitioned-models-svm.sql`
- PART_CLAS_SAMPLE by `oml4sql-partitioned-models-svm.sql`
- SVMC_SH_CLAS_SAMPLE by `oml4sql-classification-svm.sql`
- SVMO_SH_CLAS_SAMPLE by `oml4sql-anomaly-detection-1class-svm.sql`
- T_SVM_CLAS_SAMPLE by `oml4sql-classification-text-mining-svm.sql`
- SVMR_SH_REGR_SAMPLE by `oml4sql-regression-svm.sql`

34.2.2 ALL_MINING_MODEL_ATTRIBUTES

Describes an example of `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query.

The following example describes `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query. Attributes are the predictors or conditions that are used to create models and score data.

Example 34-2 ALL_MINING_MODEL_ATTRIBUTES

```
describe ALL_MINING_MODEL_ATTRIBUTES
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 ATTRIBUTE_NAME                            NOT NULL VARCHAR2(128)
 ATTRIBUTE_TYPE                                     VARCHAR2(11)
```

```
 DATA_TYPE                                          VARCHAR2(106)
 DATA_LENGTH                                        NUMBER
 DATA_PRECISION                                     NUMBER
 DATA_SCALE                                         NUMBER
 USAGE_TYPE                                         VARCHAR2(8)
 TARGET                                             VARCHAR2(3)
 ATTRIBUTE_SPEC                                     VARCHAR2(4000)
```

The following query returns the attributes of an SVM classification model named T_SVM_CLAS_SAMPLE. The model has both categorical and numerical attributes and includes one attribute that is unstructured text. The model is created by the `oml4sql-classification-text-mining-svm.sql` example

```
SELECT attribute_name, attribute_type, target
    FROM all_mining_model_attributes
    WHERE model_name = 'T_SVM_CLAS_SAMPLE'
    ORDER BY attribute_name;
```

The output is as follows:

```
ATTRIBUTE_NAME            ATTRIBUTE_TYPE
TAR
------------------------- --------------------
---
AFFINITY_CARD             CATEGORICAL
YES
AGE                       NUMERICAL
NO
BOOKKEEPING_APPLICATION   NUMERICAL
NO
BULK_PACK_DISKETTES       NUMERICAL
NO
COMMENTS                  TEXT
NO
COUNTRY_NAME              CATEGORICAL
NO
CUST_GENDER               CATEGORICAL
NO
CUST_INCOME_LEVEL         CATEGORICAL
NO
CUST_MARITAL_STATUS       CATEGORICAL
NO
EDUCATION                 CATEGORICAL
NO
FLAT_PANEL_MONITOR        NUMERICAL
NO
HOME_THEATER_PACKAGE      NUMERICAL
NO
HOUSEHOLD_SIZE            CATEGORICAL
NO
OCCUPATION                CATEGORICAL
NO
OS_DOC_SET_KANJI          NUMERICAL
```

```
NO
PRINTER_SUPPLIES          NUMERICAL
NO
YRS_RESIDENCE             NUMERICAL
NO
Y_BOX_GAMES               NUMERICAL             NO
```

34.2.3 ALL_MINING_MODEL_PARTITIONS

Describes an example of `ALL_MINING_MODEL_PARTITIONS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_PARTITIONS` and shows a sample query.

Example 34-3 ALL_MINING_MODEL_PARTITIONS

```
describe ALL_MINING_MODEL_PARTITIONS
```

The output is as follows:

```
Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 PARTITION_NAME                                     VARCHAR2(128)
 POSITION                                           NUMBER
 COLUMN_NAME                               NOT NULL VARCHAR2(128)
 COLUMN_VALUE                                       VARCHAR2(4000)
```

The following query returns the partition names and partition key values for two partitioned models. Model PART2_CLAS_SAMPLE has a two column partition key with system-generated partition names. The models are created by the `oml4sql-partitioned-models-svm.sql` example.

```
SELECT model_name, partition_name, position, column_name, column_value
    FROM all_mining_model_partitions
    ORDER BY model_name, partition_name, position;
```

The output is as follows:

```
MODEL_NAME           PARTITION_ POSITION COLUMN_NAME
COLUMN_VALUE
-------------------- ---------- -------- --------------------
---------------
PART2_CLAS_SAMPLE    DM$$_P0           1 CUST_GENDER
F
PART2_CLAS_SAMPLE    DM$$_P0           2 CUST_INCOME_LEVEL
HIGH
PART2_CLAS_SAMPLE    DM$$_P1           1 CUST_GENDER
F
```

```
PART2_CLAS_SAMPLE     DM$$_P1            2 CUST_INCOME_LEVEL
LOW
PART2_CLAS_SAMPLE     DM$$_P2            1 CUST_GENDER
F
PART2_CLAS_SAMPLE     DM$$_P2            2 CUST_INCOME_LEVEL
MEDIUM
PART2_CLAS_SAMPLE     DM$$_P3            1 CUST_GENDER
M
PART2_CLAS_SAMPLE     DM$$_P3            2 CUST_INCOME_LEVEL
HIGH
PART2_CLAS_SAMPLE     DM$$_P4            1 CUST_GENDER
M
PART2_CLAS_SAMPLE     DM$$_P4            2 CUST_INCOME_LEVEL
LOW
PART2_CLAS_SAMPLE     DM$$_P5            1 CUST_GENDER
M
PART2_CLAS_SAMPLE     DM$$_P5            2 CUST_INCOME_LEVEL
MEDIUM
PART_CLAS_SAMPLE      F                  1 CUST_GENDER
F
PART_CLAS_SAMPLE      M                  1 CUST_GENDER
M
PART_CLAS_SAMPLE      U                  1 CUST_GENDER           U
```

34.2.4 ALL_MINING_MODEL_SETTINGS

Describes an example of `ALL_MINING_MODEL_SETTINGS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_SETTINGS` and shows a sample query. Settings influence model behavior. Settings may be specific to an algorithm or to a machine learning technique, or they may be general.

Example 34-4 ALL_MINING_MODEL_SETTINGS

```
 describe ALL_MINING_MODEL_SETTINGS
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 SETTING_NAME                              NOT NULL VARCHAR2(30)
 SETTING_VALUE                                      VARCHAR2(4000)
 SETTING_TYPE                                       VARCHAR2(7)
```

The following query returns the settings for a model named SVD_SH_SAMPLE. The model uses the Singular Value Decomposition algorithm for feature extraction. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

```
SELECT setting_name, setting_value, setting_type
    FROM all_mining_model_settings
```

```
    WHERE model_name = 'SVD_SH_SAMPLE'
    ORDER BY setting_name;
```

The output is as follows:

```
SETTING_NAME                   SETTING_VALUE
SETTING
------------------------------ ------------------------------
-------
ALGO_NAME                      ALGO_SINGULAR_VALUE_DECOMP
INPUT
ODMS_DETAILS                   ODMS_ENABLE
DEFAULT
ODMS_MISSING_VALUE_TREATMENT   ODMS_MISSING_VALUE_AUTO
DEFAULT
ODMS_SAMPLING                  ODMS_SAMPLING_DISABLE
DEFAULT
PREP_AUTO                      OFF
INPUT
SVDS_SCORING_MODE              SVDS_SCORING_SVD
DEFAULT
SVDS_U_MATRIX_OUTPUT           SVDS_U_MATRIX_ENABLE           INPUT
```

34.2.5 ALL_MINING_MODEL_VIEWS

Describes an example of `ALL_MINING_MODEL_VIEWS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_VIEWS` and shows a sample query. Model views provide details on the models.

Example 34-5 ALL_MINING_MODEL_VIEWS

```
describe ALL_MINING_MODEL_VIEWS
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 VIEW_NAME                                 NOT NULL VARCHAR2(128)
 VIEW_TYPE                                          VARCHAR2(128)
```

The following query returns the model views for the SVD_SH_SAMPLE model. The model uses the Singular Value Decomposition algorithm for feature extraction. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

```
SELECT view_name, view_type
    FROM all_mining_model_views
    WHERE model_name = 'SVD_SH_SAMPLE'
    ORDER BY view_name;
```

The output is as follows:

```
VIEW_NAME
VIEW_TYPE
-------------------------
--------------------------------------------------
DM$VESVD_SH_SAMPLE        Singular Value Decomposition S
Matrix
DM$VGSVD_SH_SAMPLE        Global Name-Value
Pairs
DM$VNSVD_SH_SAMPLE        Normalization and Missing Value
Handling
DM$VSSVD_SH_SAMPLE        Computed
Settings
DM$VUSVD_SH_SAMPLE        Singular Value Decomposition U
Matrix
DM$VVSVD_SH_SAMPLE        Singular Value Decomposition V
Matrix
DM$VWSVD_SH_SAMPLE        Model Build Alerts
```

34.2.6 ALL_MINING_MODEL_XFORMS

Describes an example of `ALL_MINING_MODEL_XFORMS` and provides a sample query.

The following example describes `ALL_MINING_MODEL_XFORMS` and provides a sample query.

Example 34-6   ALL_MINING_MODEL_XFORMS

```
describe ALL_MINING_MODEL_XFORMS


 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 ATTRIBUTE_NAME                                     VARCHAR2(128)
 ATTRIBUTE_SUBNAME                                  VARCHAR2(4000)
 ATTRIBUTE_SPEC                                     VARCHAR2(4000)
 EXPRESSION                                         CLOB
 REVERSE                                            VARCHAR2(3)
```

The following query returns the embedded transformations for a model `PART2_CLAS_SAMPLE`. The model is created by the `oml4sql-partitioned-models-svm.sql` example.

```
SELECT attribute_name, expression
    FROM all_mining_model_xforms
    WHERE model_name = 'PART2_CLAS_SAMPLE'
    ORDER BY attribute_name;
```

The output is as follows:

```

ATTRIBUTE_NAME

-------------------------

EXPRESSION

-----------------------------------------------------------------------------
---
CUST_INCOME_LEVEL

CASE CUST_INCOME_LEVEL WHEN 'A: Below 30,000' THEN
'LOW'
    WHEN 'L: 300,000 and above' THEN
'HIGH'
    ELSE 'MEDIUM' END
```

34.3 Oracle Machine Learning Modeling, Transformations, and Convenience Functions

You can access PL/SQL interface to perform data modeling, transformations, and predictive analytics.

The following table displays the PL/SQL packages for Oracle Machine Learning. In Oracle Database releases prior to Release 21c, Oracle Machine Learning was named Oracle Data Mining.

Table 34-2 Oracle Machine Learning PL/SQL Packages

Package Name	Description
`DBMS_DATA_MINING`	Routines for creating and managing machine learning models
`DBMS_DATA_MINING_TRANSFORM`	Routines for transforming the data for machine learning
`DBMS_PREDICTIVE_ANALYTICS`	Routines that perform predictive analytics

Related Topics

- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_PREDICTIVE_ANALYTICS

34.3.1 DBMS_DATA_MINING

The `DBMS_DATA_MINING` package contains routines for creating machine learning models, for performing operations on the models, and for querying them.

The package includes routines for:

- Creating, dropping, and performing other DDL operations on machine learning models
- Obtaining detailed information about model attributes, rules, and other information internal to the model (model details)
- Computing test metrics for classification models
- Specifying costs for classification models
- Exporting and importing models
- Building models using Oracle Machine Learning native algorithms as well as algorithms written in R

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

34.3.2 DBMS_DATA_MINING_TRANSFORM

The `DBMS_DATA_MINING_TRANSFORM` package contains routines that perform data transformations such as binning, normalization, and outlier treatment.

The package includes routines for:

- Specifying transformations in a format that can be embedded in a machine learning model.
- Specifying transformations as relational views (external to machine learning model objects).
- Specifying distinct properties for columns in the build data. For example, you can specify that the column must be interpreted as unstructured text, or that the column must be excluded from Automatic Data Preparation.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

34.3.2.1 Transformation Methods in DBMS_DATA_MINING_TRANSFORM

Summarizes the methods for transforming data in `DBMS_DATA_MINING_TRANSFORM` package.

Table 34-3 DBMS_DATA_MINING_TRANSFORM Transformation Methods

Transformation Method	Description
`XFORM` interface	`CREATE`, `INSERT`, and `XFORM` routines specify transformations in external views

Table 34-3 (Cont.) DBMS_DATA_MINING_TRANSFORM Transformation Methods

Transformation Method	Description
STACK interface	CREATE, INSERT, and XFORM routines specify transformations for embedding in a model
SET_TRANSFORM	Specifies transformations for embedding in a model

The statements in the following example create a Support Vector Machine (SVM) classification model called T_SVM_Clas_sample with an embedded transformation that causes the comments attribute to be treated as unstructured text data. The T_SVM_CLAS_SAMPLE model is created by `oml4sql-classification-text-mining-svm.sql` example.

Example 34-7 Sample Embedded Transformation

```
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(
    xformlist, 'comments', null, 'comments', null, 'TEXT');
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name           => 'T_SVM_Clas_sample',
    mining_function      => dbms_data_mining.classification,
    data_table_name      => 'mining_build_text',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 't_svmc_sample_settings',
    xform_list => xformlist);
END;
/
```

34.3.3 DBMS_PREDICTIVE_ANALYTICS

The `DBMS_PREDICTIVE_ANALYTICS` package contains routines that perform an automated form of machine learning known as predictive analytics. With predictive analytics, you do not need to be aware of model building or scoring. All machine learning activities are handled internally by the procedure.

The `DBMS_PREDICTIVE_ANALYTICS` package includes these routines:

- **`EXPLAIN`** ranks attributes in order of influence in explaining a target column.
- **`PREDICT`** predicts the value of a target column based on values in the input data.
- **`PROFILE`** generates rules that describe the cases from the input data.

The `EXPLAIN` statement in the following example lists attributes in the view `mining_data_build_v` in order of their importance in predicting `affinity_card`.

Example 34-8 Sample EXPLAIN Statement

```
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'mining_data_build_v',
        explain_column_name  => 'affinity_card',
        result_table_name    => 'explain_results');
```

```
END;
/
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

34.4 Oracle Machine Learning for SQL Scoring Functions

Use OML4SQL functions score data. Functions can apply a machine learning model schema object to data or dynamically mine it with an analytic clause. SQL functions exist for all OML4SQL scoring algorithms.

All OML4SQL functions, as listed in the following table can operate on an R machine learning model with the corresponding OML4SQL function. However, the functions are not limited to the ones listed here.

Table 34-4 OML4SQL Functions

Function	Description
`CLUSTER_ID`	Returns the ID of the predicted cluster
`CLUSTER_DETAILS`	Returns detailed information about the predicted cluster
`CLUSTER_DISTANCE`	Returns the distance from the centroid of the predicted cluster
`CLUSTER_PROBABILITY`	Returns the probability of a case belonging to a given cluster
`CLUSTER_SET`	Returns a list of all possible clusters to which a given case belongs along with the associated probability of inclusion
`FEATURE_COMPARE`	Compares two similar and dissimilar set of texts from two different documents or keyword phrases or a combination of both
`FEATURE_ID`	Returns the ID of the feature with the highest coefficient value
`FEATURE_DETAILS`	Returns detailed information about the predicted feature
`FEATURE_SET`	Returns a list of objects containing all possible features along with the associated coefficients
`FEATURE_VALUE`	Returns the value of the predicted feature
`ORA_DM_PARTITION_NAME`	Returns the partition names for a partitioned model

Table 34-4 (Cont.) OML4SQL Functions

Function	Description
`PREDICTION`	Returns the best prediction for the target
`PREDICTION_BOUNDS`	(GLM only) Returns the upper and lower bounds of the interval wherein the predicted values (linear regression) or probabilities (logistic regression) lie.
`PREDICTION_COST`	Returns a measure of the cost of incorrect predictions
`PREDICTION_DETAILS`	Returns detailed information about the prediction
`PREDICTION_PROBABILITY`	Returns the probability of the prediction
`PREDICTION_SET`	Returns the results of a classification model, including the predictions and associated probabilities for each case
`VECTOR_EMBEDDING`	Generates a single vector embedding for different data types

The following example shows a query that returns the results of the `CLUSTER_ID` function. The query applies the model em_sh_clus_sample, which finds groups of customers that share certain characteristics. The query returns the identifiers of the clusters and the number of customers in each cluster. The em_sh_clus_sample model is created by the `oml4sql-clustering-expectation-maximization.sql` example.

Example 34-9 CLUSTER_ID Function

```
-- -List the clusters into which the customers in this
-- -data set have been grouped.
--
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
  FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;


-- List the clusters into which the customers in this
-- data set have been grouped.
--
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;
```

The output is as follows:

```
      CLUS        CNT
---------- ----------
         9        311
         3        294
         7        215
        12        201
        17        123
        16        114
        14         86
        19         64
        15         56
        18         36
```

34.5 Oracle Machine Learning for SQL Statistical Functions

Various SQL statistical functions are available in Oracle Database to explore and analyze data.

A variety of scalable statistical functions are accessible through SQL in Oracle Database. These statistical functions are implemented as SQL functions. The SQL statistical functions can be used to compute standard univariate statistics such as `MEAN`, `MAX`, `MIN`, `MEDIAN`, `MODE`, and standard deviation on the data. Users can also perform various other statistical functions such as t-test, f-test, aggregate functions, analytic functions, or ANOVA. The functions listed in the following table are available from SQL.

Table 34-5 SQL Statistical Functions Supported by OML4SQL

Function	Description
`APPROX_COUNT`	Returns approximate count of an expression
`APPROX_SUM`	Returns approximate sum of an expression
`APPROX_RANK`	Returns approximate value in a group of values
`CORR`	Retuns the coefficient of correlation of a set of number pairs
`CORR_S`	Calculates the Spearman's rho correlation coefficient
`CORR_K`	Calculates the Kendall's tau-b correlation coefficient
`COVAR_POP`	Returns the population covariance of a set of number pairs
`COVAR_SAMP`	Returns the sample covariance of a set of number pairs.
`LAG`	`LAG` is an analytic function. It provides access to more than one row of a table at the same time without a self join.
`LEAD`	`LEAD` is an analytic function. It provides access to more than one row of a table at the same time without a self join.

Table 34-5 (Cont.) SQL Statistical Functions Supported by OML4SQL

Function	Description
STATS_BINOMIAL_TEST	STATS_BINOMIAL_TEST is an exact probability test used for dichotomous variables, where only two possible values exist.
STATS_CROSSTAB	STATS_CROSSTAB is a method used to analyze two nominal variables.
STATS_F_TEST	STATS_F_TEST tests whether two variances are significantly different.
STATS_KS_TEST	STATS_KS_TEST is a Kolmogorov-Smirnov function that compares two samples to test whether they are from the same population or from populations that have the same distribution.
STATS_MODE	Takes as its argument a set of values and returns the value that occurs with the greatest frequency
STATS_MW_TEST	A Mann Whitney test compares two independent samples to test the null hypothesis that two populations have the same distribution function against the alternative hypothesis that the two distribution functions are different.
STATS_ONE_WAY_ANOVA	Tests differences in means (for groups or variables) for statistical significance by comparing two different estimates of variance
STATS_T_TEST_*	The t-test measures the significance of a difference of means
STATS_T_TEST_ONE	A one-sample t-test
STATS_T_TEST_PAIRED	A two-sample, paired t-test (also known as a crossed t-test)
STATS_T_TEST_INDEP and STATS_T_TEST_INDEPU	A t-test of two independent groups with the same variance (pooled variances) A t-test of two independent groups with unequal variance (unpooled variances)
STDDEV	returns the sample standard deviation of a set of numbers
STDDEV_POP	Computes the population standard deviation and returns the square root of the population variance
STDDEV_SAMP	Computes the cumulative sample standard deviation and returns the square root of the sample variance
SUM	Returns the sum of values

DBMS_STAT_FUNCS PL/SQL package is also available for users.

35
Prepare the Data

Learn how to access and treat the data that can be used to build a model.

- Data Requirements
- About Attributes
- Use Nested Data
- Use Market Basket Data
- Use Retail Data for Analysis
- Handle Missing Values

35.1 Data Requirements

Understand how data is stored and viewed for Oracle Machine Learning.

Machine learning activities require data that is defined within a single table or view. The information for each record must be stored in a separate row. The data records are commonly called **cases**. Each case can optionally be identified by a unique **case ID**. The table or view itself can be referred to as a **case table**.

The `CUSTOMERS` table in the `SH` schema is an example of a table that could be used for machine learning. All the information for each customer is contained in a single row. The case ID is the `CUST_ID` column. The rows listed in the following example are selected from `SH.CUSTOMERS`.

> **Note:**
>
> Oracle Machine Learning requires single-record case data for all types of models except association models, which can be built on native transactional data.

Example 35-1 Sample Case Table

```
SQL> select cust_id, cust_gender, cust_year_of_birth,
          cust_main_phone_number from sh.customers where cust_id < 11;
```

The output is as follows:

```
CUST_ID CUST_GENDER CUST_YEAR_OF_BIRTH CUST_MAIN_PHONE_NUMBER
------- ----------- ---- ------------- -------------------------
1        M              1946         127-379-8954
2        F              1957         680-327-1419
3        M              1939         115-509-3391
4        M              1934         577-104-2792
5        M              1969         563-667-7731
```

```
6         F                 1925          682-732-7260
7         F                 1986          648-272-6181
8         F                 1964          234-693-8728
9         F                 1936          697-702-2618
10        F                 1947          601-207-4099
```

Related Topics

- Use Market Basket Data
  Understand the use of association and Apriori for market basket analysis.

35.1.1 Column Data Types

Understand the different types of column data in a case table.

The columns of the case table hold the attributes that describe each case. In Example 35-1, the attributes are: `CUST_GENDER`, `CUST_YEAR_OF_BIRTH`, and `CUST_MAIN_PHONE_NUMBER`. The attributes are the predictors in a supervised model or the descriptors in an unsupervised model. The case ID, `CUST_ID`, can be viewed as a special attribute; it is not a predictor or a descriptor.

OML4SQL supports standard Oracle data types except `DATE`, `TIMESTAMP`, `RAW`, and `LONG`. Oracle Machine Learning supports date type (datetime, date, timestamp) for case_id, `CLOB`/`BLOB`/`FILE` that are interpreted as text columns, and the following collection types as well:

```
DM_NESTED_CATEGORICALS
DM_NESTED_NUMERICALS
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
```

> **Note:**
>
> The attributes with the data type `BOOLEAN` are treated as numeric with the following values: `TRUE` means `1`, `FALSE` means `0`, and `NULL` is interpreted as an unknown value. The `CASE_ID_COLUMN_NAME` attribute does not support `BOOLEAN` data type.

Related Topics

- Use Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- About Unstructured Text
  Unstructured text may contain important information that is critical to the success of a business.
- *Oracle Database SQL Language Reference*

35.1.2 Data Sets for Classification and Regression

Understand how data sets are used for training and testing the model.

You need two case tables to build and validate classification and regression models. One set of rows is used for training the model, another set of rows is used for testing the model. It is often convenient to derive the build data and test data from the same data set. For example, you could randomly select 60% of the rows for training the model; the remaining 40% could be used for testing the model.

Models that implement other machine learning functions, such as attribute importance, clustering, association, or feature extraction, do not use separate test data.

35.1.3 Scoring Requirements

Learn how scoring is done in Oracle Machine Learning for SQL.

Most machine learning models can be applied to separate data in a process known as **scoring**. Oracle Machine Learning for SQL supports the scoring operation for classification, regression, anomaly detection, clustering, and feature extraction.

The scoring process matches column names in the scoring data with the names of the columns that were used to build the model. The scoring process does not require all the columns to be present in the scoring data. If the data types do not match, OML4SQL attempts to perform type coercion. For example, if a column called `PRODUCT_RATING` is `VARCHAR2` in the training data but `NUMBER` in the scoring data, OML4SQL effectively applies a `TO_CHAR()` function to convert it.

The column in the test or scoring data must undergo the same transformations as the corresponding column in the build data. For example, if the `AGE` column in the build data was transformed from numbers to the values `CHILD`, `ADULT`, and `SENIOR`, then the `AGE` column in the scoring data must undergo the same transformation so that the model can properly evaluate it.

> **Note:**
>
> OML4SQL can embed user-specified transformation instructions in the model and reapply them whenever the model is applied. When the transformation instructions are embedded in the model, you do not need to specify them for the test or scoring data sets.
>
> OML4SQL also supports Automatic Data Preparation (ADP). When ADP is enabled, the transformations required by the algorithm are performed automatically and embedded in the model along with any user-specified transformations.

> **See Also:**
>
> Automatic Data Preparation and Embed Transformations in a Model for more information on automatic and embedded data transformations

35.2 About Attributes

Attributes are the items of data that are used in machine learning. Attributes are also referred as variables, fields, or predictors.

In predictive models, attributes are the predictors that affect a given outcome. In descriptive models, attributes are the items of information being analyzed for natural groupings or associations. For example, a table of employee data that contains attributes such as job title, date of hire, salary, age, gender, and so on.

35.2.1 Data Attributes and Model Attributes

Data attributes are columns in the data set used to build, test, or score a model. **Model attributes** are the data representations used internally by the model.

Data attributes and model attributes can be the same. For example, a column called `SIZE`, with values `S`, `M`, and `L`, are attributes used by an algorithm to build a model. Internally, the model attribute `SIZE` is most likely be the same as the data attribute from which it was derived.

On the other hand, a nested column `SALES_PROD`, containing the sales figures for a group of products, does not correspond to a model attribute. The data attribute can be `SALES_PROD`, but each product with its corresponding sales figure (each row in the nested column) is a model attribute.

Transformations also cause a discrepancy between data attributes and model attributes. For example, a transformation can apply a calculation to two data attributes and store the result in a new attribute. The new attribute is a model attribute that has no corresponding data attribute. Other transformations such as binning, normalization, and outlier treatment, cause the model's representation of an attribute to be different from the data attribute in the case table.

Related Topics

- Use Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

35.2.2 Target Attribute

Understand what a **target** means in machine learning and understand the different target data types.

The **target** of a supervised model is a special kind of attribute. The target column in the training data contains the historical values used to train the model. The target column in the test data contains the historical values to which the predictions are compared. The act of scoring produces a prediction for the target.

Clustering, feature extraction, association, and anomaly detection models do not use a target.

Nested columns and columns of unstructured data (such as `BFILE`, `CLOB`, or `BLOB`) cannot be used as targets.

Table 35-1 Target Data Types

Machine Learning Function	Target Data Types
Classification	VARCHAR2, CHAR NUMBER, FLOAT BINARY_DOUBLE, BINARY_FLOAT, ORA_MINING_VARCHAR2_NT BOOLEAN
Regression	NUMBER, FLOAT BINARY_DOUBLE, BINARY_FLOAT

You can query the `*_MINING_MODEL_ATTRIBUTES` view to find the target for a given model.

Related Topics

- ALL_MINING_MODEL_ATTRIBUTES
  Describes an example of `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query.
- *Oracle Database PL/SQL Packages and Types Reference*

35.2.3 Numericals, Categoricals, and Unstructured Text

Explains numeric, categorical, and unstructured text attributes.

Model attributes are numerical, categorical, or unstructured (text). Data attributes, which are columns in a case table, have Oracle data types, as described in "Column Data Types".

Numerical attributes can theoretically have an infinite number of values. The values have an implicit order, and the differences between them are also ordered. Oracle Machine Learning for SQL interprets `NUMBER`, `FLOAT`, `BINARY_DOUBLE`, `BINARY_FLOAT`, `BOOLEAN`, `DM_NESTED_NUMERICALS`, `DM_NESTED_BINARY_DOUBLES`, and `DM_NESTED_BINARY_FLOATS` as numerical.

Categorical attributes have values that identify a finite number of discrete categories or classes. There is no implicit order associated with the values. Some categoricals are binary: they have only two possible values, such as yes or no, or male or female. Other categoricals are multi-class: they have more than two values, such as small, medium, and large.

OML4SQL interprets `CHAR` and `VARCHAR2` as categorical by default, however these columns may also be identified as columns of unstructured data (text). OML4SQL interprets columns of `DM_NESTED_CATEGORICALS` as categorical. Columns of `CLOB`, `BLOB`, and `BFILE` always contain unstructured data.

The target of a classification model is categorical. (If the target of a classification model is numeric, it is interpreted as categorical.) The target of a regression model is numerical. The target of an attribute importance model is either categorical or numerical.

Related Topics

- Column Data Types
  Understand the different types of column data in a case table.
- About Unstructured Text
  Unstructured text may contain important information that is critical to the success of a business.

35.2.4 Model Signature

Learn about model signature and the data types that are considered in the build data.

The model signature is the set of data attributes that are used to build a model. Some or all of the attributes in the signature must be present for scoring. The model accounts for any missing columns on a best-effort basis. If columns with the same names but different data types are present, the model attempts to convert the data type. If extra, unused columns are present, they are disregarded.

The model signature does not necessarily include all the columns in the build data. Algorithm-specific criteria can cause the model to ignore certain columns. Other columns can be eliminated by transformations. Only the data attributes actually used to build the model are included in the signature.

The target and case ID columns are not included in the signature.

35.2.5 Scoping of Model Attribute Name

Learn about model attribute name.

The model attribute name consists of two parts: a column name, and a subcolumn name.

```
column_name[.subcolumn_name]
```

The `column_name` component is the name of the data attribute. It is present in all model attribute names. Nested attributes and text attributes also have a `subcolumn_name` component as shown in the following example.

Example 35-2 Model Attributes Derived from a Nested Column

The nested column `SALESPROD` has three rows.

```
SALESPROD(ATTRIBUTE_NAME, VALUE)
--------------------------------
((PROD1, 300),
 (PROD2, 245),
 (PROD3, 679))
```

The name of the data attribute is `SALESPROD`. Its associated model attributes are:

```
SALESPROD.PROD1
SALESPROD.PROD2
SALESPROD.PROD3
```

35.2.6 Model Details

Model details reveal information about model attributes and their treatment by the algorithm. Oracle recommends that users leverage the model detail views for the respective algorithm.

Transformation and reverse transformation expressions are associated with model attributes. Transformations are applied to the data attributes before the algorithmic processing that creates the model. Reverse transformations are applied to the model attributes after the model has been built, so that the model details are expressed in the form of the original data attributes, or as close to it as possible.

Reverse transformations support model transparency. They provide a view of the data that the algorithm is working with internally but in a format that is meaningful to a user.

Deprecated `GET_MODEL_DETAILS`

There is a separate `GET_MODEL_DETAILS` routine for each algorithm. Starting from Oracle Database 12*c* Release 2, the `GET_MODEL_DETAILS` are deprecated. Oracle recommends to use Model Detail Views for the respective algorithms.

Related Topics

- Model Detail Views

35.3 Use Nested Data

A join between the tables for one-to-many relationship is represented through nested columns.

Oracle Machine Learning for SQL requires a case table in single-record case format, with each record in a separate row. What if some or all of your data is in multi-record case format, with each record in several rows? What if you want one attribute to represent a series or collection of values, such as a student's test scores or the products purchased by a customer?

This kind of one-to-many relationship is usually implemented as a join between tables. For example, you can join your customer table to a sales table and thus associate a list of products purchased with each customer.

OML4SQL supports dimensioned data through nested columns. To include dimensioned data in your case table, create a view and cast the joined data to one of the machine learning nested table types. Each row in the nested column consists of an attribute name/value pair. OML4SQL internally processes each nested row as a separate attribute.

> **Note:**
>
> O-Cluster is the only algorithm that does not support nested data.

Related Topics

- Example: Creating a Nested Column for Market Basket Analysis
  The example shows how to define a nested column for market basket analysis.

35.3.1 Nested Object Types

Nested tables are object data types that can be used in place of other data types.

Oracle Database supports user-defined data types that make it possible to model real-world entities as objects in the database. **Collection types** are object data types for modeling multi-valued attributes. Nested tables are collection types. Nested tables can be used anywhere that other data types can be used.

OML4SQL supports the following nested object types:

```
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
```

```
DM_NESTED_NUMERICALS
DM_NESTED_CATEGORICALS
```

Descriptions of the nested types are provided in this example.

Example 35-3 OML4SQL Nested Data Types

```
describe dm_nested_binary_double
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_DOUBLE

describe dm_nested_binary_doubles
 DM_NESTED_BINARY_DOUBLES TABLE OF SYS.DM_NESTED_BINARY_DOUBLE
 Name                                      Null?    Type
 ----------------------------------------- --------
---------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_DOUBLE

describe dm_nested_binary_float
 Name                                      Null?    Type
 ----------------------------------------- --------
---------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_FLOAT

describe dm_nested_binary_floats
 DM_NESTED_BINARY_FLOATS TABLE OF SYS.DM_NESTED_BINARY_FLOAT
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_FLOAT

describe dm_nested_numerical
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              NUMBER

describe dm_nested_numericals
 DM_NESTED_NUMERICALS TABLE OF SYS.DM_NESTED_NUMERICAL
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              NUMBER

describe dm_nested_categorical
 Name                                      Null?    Type
 ----------------------------------------- --------
```

```
----------------------------
 ATTRIBUTE_NAME                                   VARCHAR2(4000)
 VALUE                                            VARCHAR2(4000)

describe dm_nested_categoricals
 DM_NESTED_CATEGORICALS TABLE OF SYS.DM_NESTED_CATEGORICAL
 Name                                     Null?    Type
 ---------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                   VARCHAR2(4000)
 VALUE                                            VARCHAR2(4000)
```

Related Topics

- *Oracle Database Object-Relational Developer's Guide*

35.3.2 Example: Transforming Transactional Data for Machine Learning

In this example, a comparison is shown for sale of products in four regions with data before transformation and then after transformation.

Example 35-4 shows data from a view of a sales table. It includes sales for three of the many products sold in four regions. This data is not suitable for machine learning at the product level because sales for each case (product), is stored in several rows.

Example 35-5 shows how this data can be transformed for machine learning. The case ID column is `PRODUCT`. `SALES_PER_REGION`, a nested column of type `DM_NESTED_NUMERICALS`, is a data attribute. This table is suitable for machine learning at the product case level, because the information for each case is stored in a single row.

Oracle Machine Learning for SQL treats each nested row as a separate model attribute, as shown in Example 35-6.

> **Note:**
>
> The presentation in this example is conceptual only. The data is not actually pivoted before being processed.

Example 35-4 Product Sales per Region in Multi-Record Case Format

```
PRODUCT   REGION         SALES
-------   --------   ----------
Prod1       NE           556432
Prod2       NE           670155
Prod3       NE             3111
.
.
Prod1       NW            90887
Prod2       NW           100999
Prod3       NW           750437
.
.
Prod1       SE            82153
```

```
Prod2        SE             57322
Prod3        SE             28938
.
.
Prod1        SW           3297551
Prod2        SW           4972019
Prod3        SW            884923
.
.
```

Example 35-5 Product Sales per Region in Single-Record Case Format

```
PRODUCT       SALES_PER_REGION
             (ATTRIBUTE_NAME, VALUE)
------       --------------------------
Prod1         ('NE' ,      556432)
              ('NW' ,       90887)
              ('SE' ,       82153)
              ('SW' ,     3297551)
Prod2         ('NE' ,      670155)
              ('NW' ,      100999)
              ('SE' ,       57322)
              ('SW' ,     4972019)
Prod3         ('NE' ,        3111)
              ('NW' ,      750437)
              ('SE' ,       28938)
              ('SW' ,      884923)
.
.
```

Example 35-6 Model Attributes Derived From SALES_PER_REGION

```
PRODUCT     SALES_PER_REGION.NE     SALES_PER_REGION.NW     SALES_PER_REGION.SE
SALES_PER_REGION.SW
-------     ------------------     -------------------     ------------------
-------------------
Prod1                  556432                 90887
82153                 3297551
Prod2                  670155                100999
57322                 4972019
Prod3                    3111                 750437
28938                  884923
.
.
```

35.4 Use Market Basket Data

Understand the use of association and Apriori for market basket analysis.

Market basket data identifies the items sold in a set of baskets or transactions. Oracle Machine Learning for SQL provides the association machine learning function for market basket analysis.

Association models use the Apriori algorithm to generate association rules that describe how items tend to be purchased in groups. For example, an association rule can assert that people who buy peanut butter are 80% likely to also buy jelly.

Market basket data is usually **transactional**. In transactional data, a case is a transaction and the data for a transaction is stored in multiple rows. OML4SQL association models can be built on transactional data or on single-record case data. The `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` settings specify whether the data for association rules is in transactional format.

> **Note:**
>
> Association models are the only type of model that can be built on native transactional data. For all other types of models, OML4SQL requires that the data be presented in single-record case format.

The Apriori algorithm assumes that the data is transactional and that it has many missing values. Apriori interprets all missing values as sparse data, and it has its own native mechanisms for handling sparse data.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information on the `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` settings.

35.4.1 Example: Creating a Nested Column for Market Basket Analysis

The example shows how to define a nested column for market basket analysis.

Association models can be built on native transactional data or on nested data. The following example shows how to define a nested column for market basket analysis.

The following SQL statement transforms this data to a column of type `DM_NESTED_NUMERICALS` in a view called `SALES_TRANS_CUST_NESTED`. This view can be used as a case table for machine learning.

```
CREATE VIEW sales_trans_cust_nested AS
             SELECT trans_id,
                     CAST(COLLECT(DM_NESTED_NUMERICAL(
                     prod_name, 1))
                     AS DM_NESTED_NUMERICALS) custprods
                  FROM sales_trans_cust
             GROUP BY trans_id;
```

This query returns two rows from the transformed data.

```
SELECT * FROM sales_trans_cust_nested
               WHERE trans_id < 101000
               AND trans_id > 100997;
```

The output is as follows:

```
TRANS_ID  CUSTPRODS(ATTRIBUTE_NAME, VALUE)
-------  ------------------------------------------------
100998   DM_NESTED_NUMERICALS
          (DM_NESTED_NUMERICAL('O/S Documentation Set - English', 1)
100999   DM_NESTED_NUMERICALS
          (DM_NESTED_NUMERICAL('CD-RW, High Speed Pack of 5', 1),
           DM_NESTED_NUMERICAL('External 8X CD-ROM', 1),
           DM_NESTED_NUMERICAL('SIMM- 16MB PCMCIAII card', 1))
```

Example 35-7 Convert to a Nested Column

The view `SALES_TRANS_CUST` provides a list of transaction IDs to identify each market basket and a list of the products in each basket.

```
describe sales_trans_cust
```

The output is as follows:

```
 Name                                                    Null?    Type
 ----------------------------------------------------- --------
----------------
 TRANS_ID                                                NOT NULL NUMBER
 PROD_NAME                                               NOT NULL
VARCHAR2(50)
 QUANTITY                                                         NUMBER
```

Related Topics

- Handle Missing Values
  Understand sparse data and missing values.

35.5 Use Retail Data for Analysis

Retail analysis often makes use of association rules and association models.

The association rules are enhanced to calculate aggregates along with rules or itemsets.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

35.5.1 Example: Calculating Aggregates

This example shows how to calculate aggregates using the customer grocery purchase and profit data.

Calculating Aggregates for Grocery Store Data

Assume a grocery store has the following data:

Table 35-2 Grocery Store Data

Customer	Item A	Item B	Item C	Item D
Customer 1	Buys (Profit $5.00)	Buys (Profit $3.20)	Buys (Profit $12.00)	NA
Customer 2	Buys (Profit $4.00)	NA	Buys (Profit $4.20)	NA
Customer 3	Buys (Profit $3.00)	Buys (Profit $10.00)	Buys (Profit $14.00)	Buys (Profit $8.00)
Customer 4	Buys (Profit $2.00)	NA	NA	Buys (Profit $1.00)

The basket of each customer can be viewed as a transaction. The manager of the store is interested in not only the existence of certain association rules, but also in the aggregated profit if such rules exist.

In this example, one of the association rules can be (A, B)=>C for customer 1 and customer 3. Together with this rule, the store manager may want to know the following:

- The total profit of item A appearing in this rule
- The total profit of item B appearing in this rule
- The total profit for consequent C appearing in this rule
- The total profit of all items appearing in the rule

For this rule, the profit for item A is $5.00 + $3.00 = $8.00, for item B the profit is $3.20 + $10.00 = $13.20, for consequent C, the profit is $12.00 + $14.00 = $26.00, for the antecedent itemset (A, B) is $8.00 + $13.20 = $21.20. For the whole rule, the profit is $21.20 + $26.00 = $47.40.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

35.6 Handle Missing Values

Understand sparse data and missing values.

Oracle Machine Learning for SQL distinguishes between **sparse data** and data that contains **random missing values**. The latter means that some attribute values are unknown. Sparse data, on the other hand, contains values that are assumed to be known, although they are not represented in the data.

A typical example of sparse data is market basket data. Out of hundreds or thousands of available items, only a few are present in an individual case (the basket or transaction). All the item values are known, but they are not all included in the basket. Present values have a quantity, while the items that are not represented are sparse (with a known quantity of zero).

OML4SQL interprets missing data as follows:

- Missing at random: Missing values in columns with a simple data type (not nested) are assumed to be missing at random.
- Sparse: Missing values in nested columns indicate sparsity.

35.6.1 Missing Values or Sparse Data?

Some real life examples are described to interpret missing values and sparse data.

The examples illustrate how Oracle Machine Learning for SQL identifies data as either sparse or missing at random.

35.6.1.1 Sparsity in a Sales Table

Understand how Oracle Machine Learning for SQL interprets missing data in nested column.

A sales table contains point-of-sale data for a group of products that are sold in several stores to different customers over a period of time. A particular customer buys only a few of the products. The products that the customer does not buy do not appear as rows in the sales table.

If you were to figure out the amount of money a customer has spent for each product, the unpurchased products have an inferred amount of zero. The value is not random or unknown; it is zero, even though no row appears in the table.

Note that the sales data is dimensioned (by product, stores, customers, and time) and are often represented as nested data for machine learning.

Since missing values in a nested column always indicate sparsity, you must ensure that this interpretation is appropriate for the data that you want to mine. For example, when trying to mine a multi-record case data set containing movie ratings from users of a large movie database, the missing ratings are unknown (missing at random), but Oracle Machine Learning for SQL treats the data as sparse and infer a rating of zero for the missing value.

35.6.1.2 Missing Values in a Table of Customer Data

When the data is not available for some attributes, those missing values are considered to be missing at random.

A table of customer data contains demographic data about customers. The case ID column is the customer ID. The attributes are age, education, profession, gender, house-hold size, and so on. Not all the data is available for each customer. Any missing values are considered to be missing at random. For example, if the age of customer 1 and the profession of customer 2 are not present in the data, that information is unknown. It does not indicate sparsity.

Note that the customer data is not dimensioned. There is a one-to-one mapping between the case and each of its attributes. None of the attributes are nested.

35.6.2 Missing Value Treatment in Oracle Machine Learning for SQL

Summarizes the treatment of missing values in OML4SQL.

Missing value treatment depends on the algorithm and on the nature of the data (categorical or numerical, sparse or missing at random). Missing value treatment is summarized in the following table.

> **Note:**
>
> OML4SQL performs the same missing value treatment whether or not you are using Automatic Data Preparation (ADP).

Table 35-3 Missing Value Treatment by Algorithm

Missing Data	EM, GLM, NMF, k-Means, SVD, SVM	DT, MDL, NB, OC	Apriori
NUMERICAL missing at random	The algorithm replaces missing numerical values with the mean. For Expectation Maximization (EM), the replacement only occurs in columns that are modeled with Gaussian distributions.	The algorithm handles missing values naturally as missing at random.	The algorithm interprets all missing data as sparse.
CATEGORICAL missing at random	Generalized Linear Model (GLM), Non-Negative Matrix Factorization (NMF), *k*-Means, and Support Vector Machine (SVM) replaces missing categorical values with the mode. Singular Value Decomposition (SVD) does not support categorical data. EM does not replace missing categorical values. EM treats NULLs as a distinct value with its own frequency count.	The algorithm handles missing values naturally as missing random.	The algorithm interprets all missing data as sparse.
NUMERICAL sparse	The algorithm replaces sparse numerical data with zeros.	O-Cluster does not support nested data and therefore does not support sparse data. Decision Tree (DT), Minimum Description Length (MDL), and Naive Bayes (NB) replace sparse numerical data with zeros.	The algorithm handles sparse data.
CATEGORICAL sparse	All algorithms except SVD replace sparse categorical data with zero vectors. SVD does not support categorical data.	O-Cluster does not support nested data and therefore does not support sparse data. DT, MDL, and NB replace sparse categorical data with the special value `DM$SPARSE`.	The algorithm handles sparse data.

35.6.3 Changing the Missing Value Treatment

Transform the missing data as sparse or missing at random.

If you want Oracle Machine Learning for SQL to treat missing data as sparse instead of missing at random or missing at random instead of sparse, transform it before building the model.

If you want missing values to be treated as sparse, but OML4SQL interprets them as missing at random, you can use a SQL function like `NVL` to replace the nulls with a value such as "NA". OML4SQL does not perform missing value treatment when there is a specified value.

If you want missing nested attributes to be treated as missing at random, you can transform the nested rows into physical attributes in separate columns — as long as the case table stays within the column limitation imposed by the Database. Fill in all of the possible attribute names, and specify them as null. Alternatively, insert rows in the nested column for all the items that are not present and assign a value such as the mean or mode to each one.

Related Topics

- *Oracle Database SQL Language Reference*

35.7 About Transformations

Understand how you can transform data by using Automatic Data Preparation (ADP) and embedded data transformation.

A transformation is a SQL expression that modifies the data in one or more columns. Data must typically undergo certain transformations before it can be used to build a model. Many Oracle Machine Learning algorithms have specific transformation requirements. Before data can be scored, it must be transformed in the same way that the training data was transformed.

Oracle Machine Learning for SQL supports ADP, which automatically implements the transformations required by the algorithm. The transformations are embedded in the model and automatically run whenever the model is applied.

If additional transformations are required, you can specify them as SQL expressions and supply them as input when you create the model. These transformations are embedded in the model as they are with ADP.

With automatic and embedded data transformation, most of the work of data preparation is handled for you. You can create a model and score multiple data sets in a few steps:

1. Identify the columns to include in the case table.
2. Create nested columns if you want to include transactional data.
3. Write SQL expressions for any transformations not handled by ADP.
4. Create the model, supplying the SQL expressions (if specified) and identifying any columns that contain text data.
5. Ensure that some or all of the columns in the scoring data have the same name and type as the columns used to train the model.

Related Topics

- Scoring Requirements
  Learn how scoring is done in Oracle Machine Learning for SQL.

> **See Also:**
>
> OML provides algorithm-specific automatic data preparation and other model building-related features

35.8 Prepare the Case Table

The first step in preparing data for machine learning is the creation of a case table.

If all the data resides in a single table and all the information for each case (record) is included in a single row (single-record case), this process is already taken care of. If the data resides in several tables, creating the data source involves the creation of a view. For the sake of simplicity, the term "case table" is used here to refer to either a table or a view.

35.8.1 Convert Column Data Types

In OML, string columns are treated as categorical, number columns as numerical, and `BOOLEAN` columns are treated as numerical. If you have a numeric column that you want to be treated as a categorical, you must convert it to a string. For example, the day number of the week.

For example, zip codes identify different postal zones; they do not imply order. If the zip codes are stored in a numeric column, they are interpreted as a numeric attribute. You must convert the data type so that the column data can be used as a categorical attribute by the model. You can do this using the `TO_CHAR` function to convert the digits 1-9 and the `LPAD` function to retain the leading 0, if there is one.

```
LPAD(TO_CHAR(ZIPCODE),5,'0')
```

The attributes with the data type `BOOLEAN` are treated as numeric with the following values: `TRUE` means `1`, `FALSE` means `0`, and `NULL` is interpreted as an unknown value. The `CASE_ID_COLUMN_NAME` attribute does not support `BOOLEAN` data type.

35.8.2 Extract Datetime Column Values

You can extract values from a datatime or interval value using the `EXTRACT` function.

The `EXTRACT` function extracts and returns the value of a specified datetime field from a datetime or interval value expression. The values that can be extracted are `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `TIMEZONE_HOUR`, `TIMEZONE_MINUTE`, `TIMEZONE_REGION`, and `TIMEZONE_ABBR`.

```
sales_tssales_tsCUST_IDTIME_STAMP

select cust_id, time_stamp,
    extract(year from time_stamp) year,
```

```
    extract(month from time_stamp) month,
    extract(day from time_stamp) day_of_month,
    to_char(time_stamp,'ww') week_of_year,
    to_char(time_stamp,'D') day_of_week,
    extract(hour from time_stamp) hour,
    extract(minute from time_stamp) minute,
    extract(second from time_stamp) second
from sales_ts
```

35.8.3 Text Transformation

Learn text processing using Oracle Machine Learning for SQL.

You can use OML4SQL to process text. Columns of text in the case table can be processed once they have undergone the proper transformation.

The text column must be in a table, not a view. The transformation process uses several features of Oracle Text; it treats the text in each row of the table as a separate document. Each document is transformed to a set of text tokens known as **terms**, which have a numeric value and a text label. The text column is transformed to a nested column of `DM_NESTED_NUMERICALS`.

35.8.4 About Business and Domain-Sensitive Transformations

Understand why you need to transform data according to business problems.

Some transformations are dictated by the definition of the business problem. For example, you want to build a model to predict high-revenue customers. Since your revenue data for current customers is in dollars you need to define what "high-revenue" means. Using some formula that you have developed from past experience, you can recode the revenue attribute into ranges Low, Medium, and High before building the model.

Another common business transformation is the conversion of date information into elapsed time. For example, date of birth can be converted to age.

Domain knowledge can be very important in deciding how to prepare the data. For example, some algorithms produce unreliable results if the data contains values that fall far outside of the normal range. In some cases, these values represent errors or unusualities. In others, they provide meaningful information.

Related Topics

- Outlier Treatment
  Understand what you must do to treat outliers.

35.8.5 Create Nested Columns

In transactional data, the information for each case is contained in multiple rows. When the data source includes transactional data (multi-record case), the transactions must be aggregated to the case level in nested columns.

An example is sales data in a star schema when machine learning at the product level. Sales is stored in many rows for a single product (the case) because the product is sold in many stores to many customers over a period of time.

See Also:

Using Nested Data for information about converting transactional data to nested columns

36
Create a Model

Explains how to create Oracle Machine Learning for SQL models and to query model details.

36.1 Before Creating a Model

Explains the preparation steps before creating a model.

Models are database schema objects that perform machine learning. The `DBMS_DATA_MINING` PL/SQL package is the API for creating, configuring, evaluating, and querying machine learning models (model details).

Before you create a model, you must decide what you want the model to do. You must identify the training data and determine if transformations are required. You can specify model settings to influence the behavior of the model behavior. The preparation steps are summarized in the following table.

Table 36-1 Preparation for Creating an Oracle Machine Learning for SQL Model

Preparation Step	Description
Choose the machine learning function	See Choose the Machine Learning Technique
Choose the algorithm	See Choose the Algorithm
Identify the build (training) data	See Prepare the Data
For classification and regression models, identify the test data	See Data Sets for Classification and Regression
Determine your data transformation strategy and create and populate a settings tables (if needed)	See Specify Model Settings

Related Topics

- About Oracle Machine Learning Models
  Machine learning models are database schema objects that perform machine learning techniques.
- DBMS_DATA_MINING
  The `DBMS_DATA_MINING` package contains routines for creating machine learning models, for performing operations on the models, and for querying them.

36.2 Automatic Data Preparation

Most algorithms require some form of data transformation. During the model build process, Oracle Machine Learning for SQL can automatically perform the transformations required by the algorithm.

You can choose to supplement the automatic transformations with additional transformations of your own, or you can choose to manage all the transformations yourself.

In calculating automatic transformations, OML4SQL uses heuristics that address the common requirements of a given algorithm. This process results in reasonable model quality in most cases.

Binning and normalization are transformations that are commonly needed by machine learning algorithms.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

36.2.1 Binning

Binning, also called discretization, is a technique for reducing the cardinality of continuous and discrete data. Binning groups related values together in bins to reduce the number of distinct values.

Binning can improve resource utilization and model build response time dramatically without significant loss in model quality. Binning can improve model quality by strengthening the relationship between attributes.

Supervised binning is a form of intelligent binning in which important characteristics of the data are used to determine the bin boundaries. In supervised binning, the bin boundaries are identified by a single-predictor decision tree that takes into account the joint distribution with the target. Supervised binning can be used for both numerical and categorical attributes.

36.2.2 Normalization

Learn about normalization.

Normalization is the most common technique for reducing the range of numerical data. Most normalization methods map the range of a single variable to another range (often 0,1).

36.2.3 How ADP Transforms the Data

The following table shows how ADP prepares the data for each algorithm.

Table 36-2 Oracle Machine Learning Algorithms With ADP

Algorithm	Machine Learning Function	Treatment by ADP
Apriori	Association rules	ADP has no effect on association rules.
CUR Matrix Decomposition	Feature selection	ADP has no effect on CUR Matrix Decomposition
Decision Tree	Classification	ADP has no effect on Decision Tree. Data preparation is handled by the algorithm.
Expectation Maximization	Clustering	Single-column (not nested) numerical columns that are modeled with Gaussian distributions are normalized. ADP has no effect on the other types of columns.
GLM	Classification and regression	Numerical attributes are normalized.
k-Means	Clustering	Numerical attributes are normalized.
MDL	Attribute importance	All attributes are binned with supervised binning.
MSET-SPRT	Classification (for anomaly detection)	Z-score normalization is performed.
Naive Bayes	Classification	All attributes are binned with supervised binning.
Neural Network	Classification and regression	Numerical attributes are normalized.
NMF	Feature extraction	Numerical attributes are normalized.
O-Cluster	Clustering	Numerical attributes are binned with a specialized form of equi-width binning, which computes the number of bins per attribute automatically. Numerical columns with all nulls or a single value are removed.
Random Forest	Classification	ADP has no effect on Random Forest. Data preparation is handled by the algorithm.
SVD	Feature extraction	Numeric attributes are centered if PCA is selected.
SVM	Classification, anomaly detection, and regression	Numerical attributes are normalized.
XG Boost	Classification and regression	ADP has no effect on XG Boost.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference*
- Part III, Algorithms, in *Oracle Machine Learning for SQL Concepts* for more information about algorithm-specific data preparation

36.3 Embed Transformations in a Model

You can specify your own transformations and embed them in a model by creating a transformation list and passing it to DBMS_DATA_MINING.CREATE_MODEL2 or DBMS_DATA_MINING.CREATE_MODEL.

The transformation instructions are embedded in the model and reapplied whenever the model is applied to new data.

The schema of how you can use xform_list to embed your transformations is shown here with CREATE_MODEL procedure.

```
DBMS_DATA_MINING.CREATE_MODEL2 (
model_name          IN VARCHAR2,
mining_function        IN VARCHAR2,
data_query          IN CLOB,
set_list            IN SETTING_LIST,
case_id_column_name       IN VARCHAR2 DEFAULT NULL,
target_column_name        IN VARCHAR2 DEFAULT NULL,
xform_list          IN TRANSFORM_LIST DEFAULT NULL);
```

```
DBMS_DATA_MINING.CREATE_MODEL(
                  model_name           IN VARCHAR2,
                  mining_function      IN VARCHAR2,
                  data_table_name      IN VARCHAR2,
                  case_id_column_name  IN VARCHAR2,
                  target_column_name   IN VARCHAR2 DEFAULT NULL,
                  settings_table_name  IN VARCHAR2 DEFAULT NULL,
                  data_schema_name     IN VARCHAR2 DEFAULT NULL,
                  settings_schema_name IN VARCHAR2 DEFAULT NULL,
                  xform_list          IN TRANSFORM_LIST DEFAULT NULL);
```

The following examples show how to create an embedded transform list with CREATE_MODEL and CREATE_MODEL2 procedures.

Here is an example with DBMS_DATA_MINING.CREATE_MODEL procedure:

```
BEGIN
DBMS_DATA_MINING.DROP_MODEL('model_sample2');
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE sett_table (SETTING_NAME  VARCHAR2(30),
                                    SETTING_VALUE VARCHAR2(4000));

BEGIN
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
('KMNS_DISTANCE','KMNS_EUCLIDEAN');
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
```

```
('PREP_AUTO','ON');
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
('KMNS_DETAILS', 'KMNS_DETAILS_ALL');
END;
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'N_TRANS_ATM', null,
'TO_CHAR(N_TRANS_ATM)', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'BANK_FUNDS', null,
'BANK_FUNDS+BANK_FUNDS+BANK_FUNDS', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'AGE', null,
'log(10,AGE+1)', 'power(10, AGE)-1');

 DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'model_sample2',
    mining_function     => dbms_data_mining.clustering,
    data_table_name     => 'INSUR_CUST_LTV',
    case_id_column_name => 'customer_id',
    settings_table_name => 'sett_table',
    xform_list          => xformlist);
END;
```

The following example shows how to create an embedded transformation using the `DBMS_DATA_MINING.CREATE_MODEL2` procedure:

```
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'N_TRANS_ATM', null,
'TO_CHAR(N_TRANS_ATM)', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'BANK_FUNDS', null,
'BANK_FUNDS+BANK_FUNDS+BANK_FUNDS', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'AGE', null,
'log(10,AGE+1)', 'power(10, AGE)-1');

  v_setlst('ALGO_NAME') := 'ALGO_KMEANS';

 DBMS_DATA_MINING.CREATE_MODEL2(
    model_name          => 'model_sample3',
    mining_function     => 'CLUSTERING',
    data_query          => 'select * from INSUR_CUST_LTV',
    set_list            => v_setlst,
    case_id_column_name => 'customer_id',
    xform_list          => xformlist);
END;
```

36.3.1 Specify Transformation Instructions for an Attribute

You can pass transformation instructions for an attribute by defining a transformation list.

A transformation list is defined as a table of transformation records. Each record (`transform_rec`) specifies the transformation instructions for an attribute.

```
TYPE transform_rec IS RECORD (
    attribute_name      VARCHAR2(30),
    attribute_subname   VARCHAR2(4000),
    expression          EXPRESSION_REC,
    reverse_expression  EXPRESSION_REC,
    attribute_spec      VARCHAR2(4000));
```

The fields in a transformation record are described in this table.

Table 36-3 Fields in a Transformation Record for an Attribute

Field	Description
`attribute_name` and `attribute_subname`	These fields identify the attribute, as described in "Scoping of Model Attribute Name"
`expression`	A SQL expression for transforming the attribute. For example, this expression transforms the age attribute into two categories: child and adult:[0,19) for 'child' and [19,) for adult `CASE WHEN age < 19 THEN 'child' ELSE 'adult'` Expression and reverse expressions are stored in `expression_rec` objects. See "Expression Records" for details.
`reverse_expression`	A SQL expression for reversing the transformation. For example, this expression reverses the transformation of the age attribute: `DECODE(age,'child','(-Inf,19)','[19,Inf)')`
`attribute_spec`	Specifies special treatment for the attribute. The `attribute_spec` field can be null or it can have one or more of these values: • `FORCE_IN` — For GLM, forces the inclusion of the attribute in the model build when the `ftr_selection_enable` setting is enabled. (`ftr_selection_enable` is disabled by default.) If the model is not using GLM, this value has no effect. `FORCE_IN` cannot be specified for nested attributes or text. • `NOPREP` — When ADP is on, prevents automatic transformation of the attribute. If ADP is not on, this value has no effect. You can specify `NOPREP` for a nested attribute, but not for an individual subname (row) in the nested attribute. • `TEXT` — Indicates that the attribute contains unstructured text. ADP has no effect on this setting. `TEXT` may optionally include subsettings `POLICY_NAME`, `TOKEN_TYPE`, and `MAX_FEATURES`. See Example 36-1 and Example 36-2.

Related Topics

- Scoping of Model Attribute Name
  Learn about model attribute name.
- Expression Records
  Example of a transformation record.

36.3.1.1 Expression Records

Example of a transformation record.

The transformation expressions in a transformation record are `expression_rec` objects.

```
TYPE expression_rec IS RECORD (
     lstmt       DBMS_SQL.VARCHAR2A,
     lb          BINARY_INTEGER DEFAULT 1,
     ub          BINARY_INTEGER DEFAULT 0);

TYPE varchar2a IS TABLE OF VARCHAR2(32767)
INDEX BY BINARY_INTEGER;
```

The `lstmt` field stores a `VARCHAR2A`, which allows transformation expressions to be very long, as they can be broken up across multiple rows of `VARCHAR2`. Use the `DBMS_DATA_MINING_TRANSFORM.SET_EXPRESSION` procedure to create an `expression_rec`.

36.3.1.2 Attribute Specifications

Learn how to define the characteristics specific to an attribute through attribute specification.

The attribute specification in a transformation record defines characteristics that are specific to this attribute. If not null, the attribute specification can include values `FORCE_IN`, `NOPREP`, or `TEXT`, as described in Table 36-3.

Example 36-1 An Attribute Specification with Multiple Keywords

If more than one attribute specification keyword is applicable, you can provide them in a comma-delimited list. The following expression is the specification for an attribute in a GLM model. Assuming that the `ftr_selection_enable` setting is enabled, this expression forces the attribute to be included in the model. If ADP is on, automatic transformation of the attribute is not performed.

```
"FORCE_IN,NOPREP"
```

Example 36-2 A Text Attribute Specification

For text attributes, you can optionally specify subsettings `POLICY_NAME`, `TOKEN_TYPE`, and `MAX_FEATURES`. The subsettings provide configuration information that is specific to text transformation. In this example, the transformation instructions for the text content are defined in a text policy named `my_policy` with token type is `THEME`. The maximum number of extracted features is 3000.

```
"TEXT(POLICY_NAME:my_policy)(TOKEN_TYPE:THEME)(MAX_FEATURES:3000)"
```

Related Topics

- Configure a Text Attribute
  Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.

36.3.2 Build a Transformation List

You can build transformation list by `SET_TRANSFORM`, `STACK`, and `GET_*` methods. These methods are listed here.

A transformation list is a collection of transformation records. When a new transformation record is added, it is appended to the top of the transformation list. You can use any of the following methods to build a transformation list:

- The `SET_TRANFORM` procedure in `DBMS_DATA_MINING_TRANSFORM`
- The `STACK` interface in `DBMS_DATA_MINING_TRANSFORM`

- The GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST functions in DBMS_DATA_MINING

36.3.2.1 SET_TRANSFORM

The SET_TRANSFORM procedure applies a specified SQL expression to a specified attribute.

The SET_TRANSFORM procedure adds a single transformation record to a transformation list.

```
DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM (
          xform_list                IN OUT NOCOPY TRANSFORM_LIST,
          attribute_name            VARCHAR2,
          attribute_subname         VARCHAR2,
          expression                VARCHAR2,
          reverse_expression        VARCHAR2,
          attribute_spec            VARCHAR2 DEFAULT NULL);
```

SQL expressions that you specify with SET_TRANSFORM must fit within a VARCHAR2. To specify a longer expression, you can use the SET_EXPRESSION procedure, which builds an expression by appending rows to a VARCHAR2 array. For example, the following statement appends a transformation instruction for country_id to a list of transformations called my_xforms. The transformation instruction divides country_id by 10 before algorithmic processing begins. The reverse transformation multiplies country_id by 10.

```
  dbms_data_mining_transform.SET_TRANSFORM (my_xforms,
     'country_id', NULL, 'country_id/10', 'country_id*10');
```

The reverse transformation is applied in the model details. If country_id is the target of a supervised model, the reverse transformation is also applied to the scored target.

36.3.2.2 The STACK Interface

The STACK interface creates transformation records from a table of transformation instructions and adds them to a transformation list.

The STACK interface offers a set of pre-defined transformations that you can apply to an attribute or to a group of attributes. For example, you can specify supervised binning for all categorical attributes.

The STACK interface specifies that all or some of the attributes of a given type must be transformed in the same way. For example, STACK_BIN_CAT appends binning instructions for categorical attributes to a transformation list. The STACK interface consists of three steps:

1. A CREATE procedure creates a transformation definition table. For example, CREATE_BIN_CAT creates a table to hold categorical binning instructions. The table has columns for storing the name of the attribute, the value of the attribute, and the bin assignment for the value.
2. An INSERT procedure computes the bin boundaries for one or more attributes and populates the definition table. For example, INSERT_BIN_CAT_FREQ performs frequency-based binning on some or all of the categorical attributes in the data source and populates a table created by CREATE_BIN_CAT.

3. A `STACK` procedure creates transformation records from the information in the definition table and appends the transformation records to a transformation list. For example, `STACK_BIN_CAT` creates transformation records for the information stored in a categorical binning definition table and appends the transformation records to a transformation list.

36.3.2.3 GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST

Use the functions to create a new transformation list.

These two functions can be used to create a new transformation list from the transformations embedded in an existing model.

The `GET_MODEL_TRANSFORMATIONS` function returns a list of embedded transformations.

```
DBMS_DATA_MINING.GET_MODEL_TRANSFORMATIONS (
      model_name      IN VARCHAR2)
RETURN DM_TRANSFORMS PIPELINED;
```

`GET_MODEL_TRANSFORMATIONS` returns a table of `dm_transform` objects. Each `dm_transform` has these fields

```
attribute_name       VARCHAR2(4000)
attribute_subname    VARCHAR2(4000)
expression           CLOB
reverse_expression   CLOB
```

The components of a transformation list are `transform_rec`, not `dm_transform`. The fields of a `transform_rec` are described in Table 36-3. You can call `GET_MODEL_TRANSFORMATIONS` to convert a list of `dm_transform` objects to `transform_rec` objects and append each `transform_rec` to a transformation list.

```
DBMS_DATA_MINING.GET_TRANSFORM_LIST (
      xform_list           OUT NOCOPY TRANSFORM_LIST,
      model_xforms         IN   DM_TRANSFORMS);
```

> **See Also:**
>
> "DBMS_DATA_MINING_TRANSFORM Operational Notes", "SET_TRANSFORM Procedure", "CREATE_MODEL Procedure", and "GET_MODEL_TRANSFORMATIONS Function" in *Oracle Database PL/SQL Packages and Types Reference*

36.3.3 Transformation Lists and Automatic Data Preparation

You can use Automatic Data Preparation (ADP) and transformation lists to customize the data transformation.

If you enable ADP and you specify a transformation list, the transformation list is embedded with the automatic, system-generated transformations. The transformation list is processed before the automatic transformations.

If you enable ADP and do not specify a transformation list, only the automatic transformations are embedded in the model.

If ADP is disabled (the default) and you specify a transformation list, your custom transformations are embedded in the model. No automatic transformations are performed.

If ADP is disabled (the default) and you do not specify a transformation list, no transformations is embedded in the model. You have to transform the training, test, and scoring data sets yourself if necessary. You must take care to apply the same transformations to each data set.

36.3.4 Oracle Machine Learning for SQL Transformation Routines

Learn about transformation routines.

OML4SQL provides routines that implement various transformation techniques in the `DBMS_DATA_MINING_TRANSFORM` package.

Related Topics

- *Oracle Database SQL Language Reference*

36.3.4.1 Binning Routines

Explains binning techniques in OML4SQL.

A number of factors go into deciding a binning strategy. Having fewer values typically leads to a more compact model and one that builds faster, but it can also lead to some loss in accuracy.

Model quality can improve significantly with well-chosen bin boundaries. For example, an appropriate way to bin ages is to separate them into groups of interest, such as children 0-13, teenagers 13-19, youth 19-24, working adults 24-35, and so on.

The following table lists the binning techniques provided by OML4SQL:

Table 36-4 Binning Methods in DBMS_DATA_MINING_TRANSFORM

Binning Method	Description
Top-N Most Frequent Items	You can use this technique to bin categorical attributes. You specify the number of bins. The value that occurs most frequently is labeled as the first bin, the value that appears with the next frequency is labeled as the second bin, and so on. All remaining values are in an additional bin.
Supervised Binning	Supervised binning is a form of intelligent binning, where bin boundaries are derived from important characteristics of the data. Supervised binning builds a single-predictor decision tree to find the interesting bin boundaries with respect to a target. It can be used for numerical or categorical attributes.
Equi-Width Binning	You can use equi-width binning for numerical attributes. The range of values is computed by subtracting the minimum value from the maximum value, then the range of values is divided into equal intervals. You can specify the number of bins or it can be calculated automatically. Equi-width binning must usually be used with outlier treatment.

Table 36-4 (Cont.) Binning Methods in DBMS_DATA_MINING_TRANSFORM

Binning Method	Description
Quantile Binning	Quantile binning is a numerical binning technique. Quantiles are computed using the SQL analytic function `NTILE`. The bin boundaries are based on the minimum values for each quantile. Bins with equal left and right boundaries are collapsed, possibly resulting in fewer bins than requested.

Related Topics

- Routines for Outlier Treatment
  Understand the transformations used for outlier treatment.

36.3.4.2 Normalization Routines

Learn about normalization routines in Oracle Machine Learning for SQL.

Most normalization methods map the range of a single attribute to another range, typically 0 to 1 or -1 to +1.

Normalization is very sensitive to outliers. Without outlier treatment, most values are mapped to a tiny range, resulting in a significant loss of information.

Table 36-5 Normalization Methods in DBMS_DATA_MINING_TRANSFORM

Transformation	Description
Min-Max Normalization	This technique computes the normalization of an attribute using the minimum and maximum values. The shift is the minimum value, and the scale is the difference between the maximum and minimum values.
Scale Normalization	This normalization technique also uses the minimum and maximum values. For scale normalization, shift = 0, and scale = max{abs(max), abs(min)}.
Z-Score Normalization	This technique computes the normalization of an attribute using the mean and the standard deviation. Shift is the mean, and scale is the standard deviation.

Related Topics

- Routines for Outlier Treatment
  Understand the transformations used for outlier treatment.

36.3.4.3 Outlier Treatment

Understand what you must do to treat outliers.

A value is considered an outlier if it deviates significantly from most other values in the column. The presence of outliers can have a skewing effect on the data and can interfere with the effectiveness of transformations such as normalization or binning.

Outlier treatment methods such as trimming or clipping can be implemented to minimize the effect of outliers.

Outliers represent problematic data, for example, a bad reading due to the unusual condition of an instrument. However, in some cases, especially in the business arena, outliers are perfectly valid. For example, in census data, the earnings for some of the richest individuals can vary significantly from the general population. Do not treat this information as an outlier, since it is an important part of the data. You need domain knowledge to determine outlier handling.

36.3.4.4 Routines for Outlier Treatment

Understand the transformations used for outlier treatment.

Outliers are extreme values, typically several standard deviations from the mean. To minimize the effect of outliers, you can Winsorize or trim the data.

Winsorizing involves setting the tail values of an attribute to some specified value. For example, for a 90% Winsorization, the bottom 5% of values are set equal to the minimum value in the 5th percentile, while the upper 5% of values are set equal to the maximum value in the 95th percentile.

Trimming sets the tail values to NULL. The algorithm treats them as missing values.

Outliers affect the different algorithms in different ways. In general, outliers cause distortion with equi-width binning and min-max normalization.

Table 36-6 Outlier Treatment Methods in DBMS_DATA_MINING_TRANSFORM

Transformation	Description
Trimming	This technique trims the outliers in numeric columns by sorting the non-null values, computing the tail values based on some fraction, and replacing the tail values with nulls.
Windsorizing	This technique trims the outliers in numeric columns by sorting the non-null values, computing the tail values based on some fraction, and replacing the tail values with some specified value.

36.4 Understand Reverse Transformations

Reverse transformations ensure that information returned by the model is expressed in a format that is similar to or the same as the format of the data that was used to train the model. Internal transformation are reversed in the model details and in the results of scoring.

Some of the attributes used by the model correspond to columns in the build data. However, because of logic specific to the algorithm, nested data, and transformations, some attributes do not correspond to columns.

For example, a nested column in the training data is not interpreted as an attribute by the model. During the model build,OML4SQL explodes nested columns, and each row (an attribute name/value pair) becomes an attribute.

Some algorithms, for example Support Vector Machine (SVM) and Generalized Linear Model (GLM), only operate on numeric attributes. Any non-numeric column in the build data is exploded into binary attributes, one for each distinct value in the column (SVM). GLM does not generate a new attribute for the most frequent value in the original column. These binary attributes are set to one only if the column value for the case is equal to the value associated with the binary attribute.

Algorithms that generate coefficients present challenges in interpreting the results. Examples are SVM and Non-Negative Matrix Factorization (NMF). These algorithms produce coefficients that are used in combination with the transformed attributes. The coefficients are relevant to the data on the transformed scale, not the original data scale.

For all these reasons, the attributes listed in the model details do not resemble the columns of data used to train the model. However, attributes that undergo embedded transformations, whether initiated by Automatic Data Preparation (ADP) or by a user-specified transformation list, appear in the model details in their pre-transformed state, as close as possible to the original column values. Although the attributes are transformed when they are used by the model, they are visible in the model details in a form that can be interpreted by a user.

Related Topics

- ALTER_REVERSE_EXPRESSION Procedure
- GET_MODEL_TRANSFORMATIONS Function
- Model Detail Views

36.5 The CREATE_MODEL Procedure

The `CREATE_MODEL` procedure of the `DBMS_DATA_MINING` package uses the specified data to create a machine learning model with the specified name and machine learning function.

The model can be created with configuration settings and user-specified transformations.

```
PROCEDURE CREATE_MODEL(
                  model_name            IN VARCHAR2,
                  mining_function       IN VARCHAR2,
                  data_table_name       IN VARCHAR2,
                  case_id_column_name   IN VARCHAR2,
                  target_column_name    IN VARCHAR2 DEFAULT NULL,
                  settings_table_name   IN VARCHAR2 DEFAULT NULL,
                  data_schema_name      IN VARCHAR2 DEFAULT NULL,
                  settings_schema_name  IN VARCHAR2 DEFAULT NULL,
                  xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

You can also rename the model using the `RENAME_MODEL` procedure of the `DBMS_DATA_MINING` package. The procedure changes the value of the machine learning model specified against `MODEL_NAME` with another name that you specify.

The following example builds a classification model using the Support Vector Machine algorithm.

```
 Create the settings table
CREATE TABLE svm_model_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(30));

-- Populate the settings table
-- Specify SVM. By default, Naive Bayes is used for classification.
-- Specify ADP. By default, ADP is not used.
BEGIN
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
  COMMIT;
END;
```

```
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svm_model_settings');
END;
/
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- RENAME_MODEL Procedure

36.5.1 Choose the Machine Learning Technique

Describes providing an Oracle Machine Learning for SQL machine learning function for the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

An OML4SQL machine learning technique specifies a class of problems that can be modeled and solved. You specify a machine learning with the `mining_function` argument of the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

OML4SQL machine learning functions implement either **supervised** or **unsupervised** learning. Supervised learning uses a set of independent attributes to predict the value of a dependent attribute or **target**. Unsupervised learning does not distinguish between dependent and independent attributes. Supervised functions are predictive. Unsupervised functions are descriptive.

Note:

In OML4SQL terminology, a **function** is a general type of problem to be solved by a given approach to machine learning. In SQL language terminology, a **function** is an operation that returns a result.

In OML4SQL documentation, the term **function**, or **machine learning function** refers to an OML4SQL machine learning function; the term **SQL function** or **SQL machine learning function** refers to a SQL function for scoring (applying machine learning models).

You can specify any of the values in the following table for the *`mining_function`* parameter to the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

Table 36-7 Oracle Machine Learning mining_function Values

mining_function **Value**	**Description**
`ASSOCIATION`	Association is a descriptive machine learning function. An association model identifies relationships and the probability of their occurrence within a data set (association rules). Association models use the Apriori algorithm.
`ATTRIBUTE_IMPORTANCE`	Attribute importance is a predictive machine learning function. An attribute importance model identifies the relative importance of attributes in predicting a given outcome. Attribute importance models use the Minimum Description Length algorithm and CUR Matrix Decomposition.
`CLASSIFICATION`	Classification is a predictive machine learning function. A classification model uses historical data to predict a categorical target. Classification models can use Naive Bayes, Neural Network, Decision Tree, logistic regression, Random Forest, Support Vector Machine, Explicit Semantic Analysis, or XGBoost. The default is Naive Bayes. You can also specify the classification machine learning function for anomaly detection for a One-Class SVM model and a Multivariate State Estimation Technique - Sequential Probability Ratio Test model.
`CLUSTERING`	Clustering is a descriptive machine learning function. A clustering model identifies natural groupings within a data set. Clustering models can use *k*-Means, O-Cluster, or Expectation Maximization. The default is *k*-Means.
`FEATURE_EXTRACTION`	Feature extraction is a descriptive machine learning function. A feature extraction model creates a set of optimized attributes. Feature extraction models can use Non-Negative Matrix Factorization, Singular Value Decomposition (which can also be used for Principal Component Analysis) or Explicit Semantic Analysis. The default is Non-Negative Matrix Factorization.
`REGRESSION`	Regression is a predictive machine learning function. A regression model uses historical data to predict a numerical target. Regression models can use Support Vector Machine, GLM regression, or XGBoost. The default is Support Vector Machine.
`TIME_SERIES`	Time series is a predictive machine learning function. A time series model forecasts the future values of a time-ordered series of historical numeric data over a user-specified time window. Time series models use the Exponential Smoothing algorithm. The default is Exponential Smoothing.

36.5.2 Choose the Algorithm

Learn about providing the algorithm settings for a model.

The `ALGO_NAME` setting specifies the algorithm for a model. If you use the default algorithm for the machine learning technique, or if there is only one algorithm available for the machine learning technique, then you do not need to specify the `ALGO_NAME` setting.

Table 36-8 Oracle Machine Learning Algorithms

ALGO_NAME Value	Algorithm	Default?	Machine Learning Model Function
ALGO_AI_MDL	Minimum Description Length	—	Attribute importance
ALGO_APRIORI_ASSOCIATION_RULES	Apriori	—	Association
ALGO_CUR_DECOMPOSITION	CUR Matrix Decomposition	—	Attribute importance
ALGO_DECISION_TREE	Decision Tree	—	Classification
ALGO_EXPECTATION_MAXIMIZATION	Expectation Maximization	—	Clustering and Anomaly Detection
ALGO_EXPLICIT_SEMANTIC_ANALYS	Explicit Semantic Analysis	—	Feature extraction and classification
ALGO_EXPONENTIAL_SMOOTHING	Exponential Smoothing	—	Time series and time series regression
ALGO_EXTENSIBLE_LANG	Language used for an extensible algorithm	—	All machine learning functions are supported
ALGO_GENERALIZED_LINEAR_MODEL	Generalized Linear Model	—	Classification and regression
ALGO_KMEANS	*k*-Means	yes	Clustering
ALGO_MSET_SPRT	Multivariate State Estimation Technique - Sequential Probability Ratio Test	—	Anomaly detection (classification with no target)
ALGO_NAIVE_BAYES	Naive Bayes	yes	Classification
ALGO_NEURAL_NETWORK	Neural Network	—	Classification
ALGO_NONNEGATIVE_MATRIX_FACTOR	Non-Negative Matrix Factorization	yes	Feature extraction
ALGO_O_CLUSTER	O-Cluster	—	Clustering
ALGO_RANDOM_FOREST	Random Forest	—	Classification
ALGO_SINGULAR_VALUE_DECOMP	Singular Value Decomposition (can also be used for Principal Component Analysis)	—	Feature extraction
ALGO_SUPPORT_VECTOR_MACHINES	Support Vector Machine	yes	Default regression algorithm; regression, classification, and anomaly detection (classification with no target)
ALGO_XGBOOST	XGBoost	—	Classification and regression

36.5.3 Supply Transformations

Use `xform_list` to specify transformations in the model creation procedures.

You can optionally specify transformations for the build data in the *`xform_list`* parameter to `CREATE_MODEL2` and `CREATE_MODEL` procedures. The transformation instructions are embedded in the model and reapplied whenever the model is applied to new data.

36.5.3.1 Create a Transformation List

You can create a transformation list using the `DBMS_DATA_MINING_TRANSFORM` package.

The following are the ways to create a transformation list:

- The `STACK` interface in `DBMS_DATA_MINING_TRANSFORM`.

  The `STACK` interface offers a set of pre-defined transformations that you can apply to an attribute or to a group of attributes. For example, you can specify supervised binning for all categorical attributes.

- The `SET_TRANSFORM` procedure in `DBMS_DATA_MINING_TRANSFORM`.

  The `SET_TRANSFORM` procedure applies a specified SQL expression to a specified attribute. For example, the following statement appends a transformation instruction for `country_id` to a list of transformations called `my_xforms`. The transformation instruction divides `country_id` by 10 before algorithmic processing begins. The reverse transformation multiplies `country_id` by 10.

  ```
  dbms_data_mining_transform.SET_TRANSFORM (my_xforms,
     'country_id', NULL, 'country_id/10', 'country_id*10');
  ```

  The reverse transformation is applied in the model details. If `country_id` is the target of a supervised model, the reverse transformation is also applied to the scored target.

36.5.3.2 Transformation List and Automatic Data Preparation

You can provide transformation list and Automatic Data Preparation (ADP) to customize the data transformation.

The transformation list argument to `CREATE_MODEL2` and `CREATE_MODEL` interacts with the `PREP_AUTO` setting, which controls ADP:

- When ADP is on and you specify a transformation list, your transformations are applied with the automatic transformations and embedded in the model. The transformations that you specify are processed before the automatic transformations.
- When ADP is off and you specify a transformation list, your transformations are applied and embedded in the model, but no system-generated transformations are performed.
- When ADP is on and you do not specify a transformation list, the system-generated transformations are applied and embedded in the model.
- When ADP is off and you do not specify a transformation list, no transformations are embedded in the model; you must separately prepare the data sets you use for building, testing, and scoring the model.

Related Topics

- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.
- *Oracle Database PL/SQL Packages and Types Reference*

36.5.4 About Partitioned Models

Introduces partitioned models to organize and represent multiple models.

When you build a model on your data set and apply it to new data, sometimes the prediction may be generic that performs badly when run on new and evolving data. To overcome this, the data set can be divided into different parts based on some characteristics. Oracle Machine Learning for SQL supports partitioned model. Partitioned models allow users to build a type of ensemble model for each data partition. The top-level model has sub models that are automatically produced. The sub models are based on the attribute options. For example, if your data set has an attribute called `REGION` with four values and you have defined it as the partitioned attribute. Then, four sub models are created for this attribute. The sub models are automatically managed and used as a single model. The partitioned model automates a typical machine learning task and can potentially achieve better accuracy through multiple targeted models.

The partitioned model and its sub models reside as first class, persistent database objects. Persistent means that the partitioned model has an on-disk representation. In a partition model, the performance of partitioned models with a large number of partitions is enhanced, and dropping a single model within a partition model is also improved.

To create a partitioned model, include the `ODMS_PARTITION_COLUMNS` setting. To define the number of partitions, include the `ODMS_MAX_PARTITIONS` setting. When you are making predictions, you must use the top-level model. The correct sub model is selected automatically based on the attribute, the attribute options, and the partition setting. You must include the partition columns as part of the `USING` clause when scoring. The `GROUPING` hint is an optional hint that applies to machine learning scoring functions when scoring partitioned models.

The partition names, key values, and the structure of the partitioned model are available in the `ALL_MINING_MODEL_PARTITIONS` view.

Related Topics

- *Oracle Database Reference*

> **See Also:**
>
> *Oracle Database SQL Language Reference* on how to use `GROUPING` hint.
> *Oracle Machine Learning for SQL User's Guide* to understand more about partitioned models.

36.5.4.1 Partitioned Model Build Process

To build a partitioned model, Oracle Machine Learning for SQL requires a partitioning key specified in a settings table.

The partitioning key is a comma-separated list of one or more columns (up to 16) from the input data set. The partitioning key horizontally slices the input data based on discrete values of the partitioning key. That is, partitioning is performed as list values as opposed to range partitioning against a continuous value. The partitioning key supports only columns of the data type `NUMBER` and `VARCHAR2`.

During the build process the input data set is partitioned based on the distinct values of the specified key. Each data slice (unique key value) results in its own model partition. The resultant model partition is not separate and is not visible to you as a standalone model. The default value of the maximum number of partitions for partitioned models is 1000 partitions. You can also set a different maximum partitions value. If the number of partitions in the input data set exceeds the defined maximum, OML4SQL throws an exception.

The partitioned model organizes features common to all partitions and the partition specific features. The common features consist of the following metadata:

- The model name
- The machine learning function
- The machine learning algorithm
- A super set of all machine learning model attributes referenced by all partitions (signature)
- A common set of user-defined column transformations
- Any user-specified or default build settings that are interpreted as global; for example, the Auto Data Preparation (ADP) setting

36.5.4.2 DDL in Partitioned model

Learn about maintenance of partitioned models thorough DDL operations.

Partitioned models are maintained through the following DDL operations:

- Drop model or drop partition
- Add partition

36.5.4.2.1 Drop Model or Drop Partition

Oracle Machine Learning for SQL supports dropping a single model partition for a given partition name.

If only a single partition remains, you cannot explicitly drop that partition. Instead, you must either add additional partitions prior to dropping the partition or you may choose to drop the model itself. When dropping a partitioned model, all partitions are dropped in a single atomic operation. From a performance perspective, Oracle recommends `DROP_PARTITION` followed by an `ADD_PARTITION` instead of leveraging the `REPLACE` option due to the efficient behavior of the `DROP_PARTITION` option.

36.5.4.2.2 Add Partition

Oracle Machine Learning for SQL supports adding a single partition or multiple partitions to an existing partitioned model.

The addition occurs based on the input data set and the name of the existing partitioned model. The operation takes the input data set and the existing partitioned model as parameters. The partition keys are extracted from the input data set and the model partitions are built against the input data set. These partitions are added to the partitioned model. In the case where partition keys for new partitions conflict with the existing partitions in the model, you can select from the following three approaches to resolve the conflicts:

- `ERROR`: Terminates the ADD operation without adding any partitions.
- `REPLACE`: Replaces the existing partition for which the conflicting keys are found.
- `IGNORE`: Eliminates the rows having the conflicting keys.

If the input data set contains multiple keys, then the operation creates multiple partitions. If the total number of partitions in the model increases to more than the user-defined maximum specified when the model was created, then you get an error. The default threshold value for the number of partitions is 1000.

36.5.4.3 Partitioned Model Scoring

The scoring of the partitioned model is the same as that of the non-partitioned model.

The syntax of the machine learning function remains the same but is extended to provide an optional hint. The optional hint can impact the performance of a query which involves scoring a partitioned model.

For scoring a partitioned model, the signature columns used during the build for the partitioning key must be present in the scoring data set. These columns are combined to form a unique partition key. The unique key is then mapped to a specific underlying model partition, and the identified model partition is used to score that row.

The partitioned objects that are necessary for scoring are loaded on demand during the query execution and are aged out depending on the System Global Area (SGA) memory.

In this example an SVM model is used to predict the number of years a customer resides at their residence but partitioned on customer gender. The model is then used to predict the target. This example highlights the model settings that you can define when you create a partitioned model. The following example is using a view created from the SH schema tables. The `CREATE_MODEL2` procedure is used for creating the model. The partition attribute is `CUST_GENDER`. This attribute has two options *M* and *F*.

```
%script
BEGIN DBMS_DATA_MINING.DROP_MODEL('SVM_MOD_PARTITIONED');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME'):= 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlst('SVMS_KERNEL_FUNCTION')  :='SVMS_LINEAR';
    v_setlst('ODMS_PARTITION_COLUMNS'):='CUST_GENDER';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MOD_PARTITIONED',
        MINING_FUNCTION     => 'REGRESSION',
        DATA_QUERY          => 'SELECT * FROM CUSTOMERS_DEMO',
        SET_LIST            => v_setlst,
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        TARGET_COLUMN_NAME  => 'YRS_RESIDENCE');
END;
```

The output is as follows:

```
PL/SQL procedure successfully completed.


---------------------------

PL/SQL procedure successfully completed.
```

The following code sample shows the prediction.

```
%script

SELECT cust_id, YRS_RESIDENCE,
       ROUND(PREDICTION(SVM_MOD_PARTITIONED USING *),2) pred_YRS_RESIDENCE
FROM CUSTOMERS_DEMO;



CUST_ID   YRS_RESIDENCE   PRED_YRS_RESIDENCE
   100100               4                 4.71
   100200               2                 1.62
   100300               4                 4.66
   100400               6                  5.9
   100500               2                 2.07
   100600               3                 2.74
   100700               6                 5.78
   100800               5                 7.22
   100900               4                 4.88
   101000               7                 6.49
   101100               4                 3.54
   101200               1                 1.46
   101300               4                 4.34
   101400               4                 4.34 ...
```

Related Topics

- *Oracle Database SQL Language Reference*

36.6 The CREATE_MODEL2 Procedure

The `CREATE_MODEL2` procedure of the `DBMS_DATA_MINING` package is a procedure for defining model settings to build a model.

By using the `CREATE_MODEL2` procedure, the user does not need to create transient database objects. The model can use configuration settings and user-specified transformations. In the `CREATE_MODEL2` procedure, the input is a table or a view and if such an object is not already present, the user must create it.

```
DBMS_DATA_MINING.CREATE_MODEL2 (
model_name            IN VARCHAR2,
mining_function       IN VARCHAR2,
```

```
data_query            IN CLOB,
set_list              IN SETTING_LIST,
case_id_column_name   IN VARCHAR2 DEFAULT NULL,
target_column_name    IN VARCHAR2 DEFAULT NULL,
xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

The `data_query` parameter species a query which provides training data for building the model. The `set_list` parameter specifies the `SETTING_LIST`. `SETTING_LIST` is a table of CLOB index by `VARCHAR2(30)`; Where the index is the setting name and the CLOB is the setting value for that name. The rest of the parameters are covered in the `CREATE_MODEL` procedure.

You can also rename the model using the `RENAME_MODEL` procedure of the `DBMS_DATA_MINING` package. The procedure changes the value of the machine learning model specified against `MODEL_NAME` with another name that you specify.

The following `CREATE_MODEL2` procedure builds a classification model using SVM algorithm. The following example mining_data_build_v data set to arrive at likelihood of customers opting the affinity card program. .

```
DECLARE
    v_setlist DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlist('PREP_AUTO') := 'ON';
    v_setlist('ALGO_NAME') := 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlist('SVMS_KERNEL_FUNCTION') := 'SVMS_LINEAR';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MODEL',
        MINING_FUNCTION     => 'CLASSIFICATION',
        DATA_QUERY          => 'select * from mining_data_build_v',
        SET_LIST            => v_setlist,
        CASE_ID_COLUMN_NAME => 'CUST_ID,
    TARGET_COLUMN_NAME  => 'AFFINITY_CARD');
END;
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- RENAME_MODEL Procedure

36.7 Specify Model Settings

You can configure your model by specifying model settings.

Numerous configuration settings are available for configuring machine learning models at build time. Specify your model settings in `CREATE_MODEL` or `CREATE_MODEL2` procedures. To specify settings in `CREATE_MODEL` procedure, create a settings table with the columns shown in the following table and pass the table to in the procedure.

You can also use `CREATE_MODEL2` procedure where you can directly pass the model settings to a variable that can be used in the procedure. The variable can be declared with `DBMS_DATA_MINING.SETTING_LIST` procedure.

Table 36-9 Settings Table Required Columns

Column Name	Data Type
`setting_name`	`VARCHAR2(30)`
`setting_value`	`VARCHAR2(4000)`

Example 36-3 creates a settings table for a Support Vector Machine (SVM) classification model. Since SVM is not the default classifier, the `ALGO_NAME` setting is used to specify the algorithm. Setting the `SVMS_KERNEL_FUNCTION` to `SVMS_LINEAR` causes the model to be built with a linear kernel. If you do not specify the kernel function, the algorithm chooses the kernel based on the number of attributes in the data.

Example 36-4 creates a model with the model settings that are stored in a variable from `SETTING_LIST`.

Some settings apply generally to the model, others are specific to an algorithm. Model settings are referenced in Table 36-10 and Table 36-11.

Table 36-10 General Model Settings

Settings	Description
Machine learning function settings	Machine Learning Technique Settings
Algorithm names	Algorithm Names
Global model characteristics	Global Settings
Automatic Data Preparation	Automatic Data Preparation

Table 36-11 Algorithm-Specific Model Settings

Algorithm	Description
CUR Matrix Decomposition	DBMS_DATA_MINING —Algorithm Settings: CUR Matrix Decomposition
Decision Tree	DBMS_DATA_MINING —Algorithm Settings: Decision Tree
Expectation Maximization	DBMS_DATA_MINING —Algorithm Settings: Expectation Maximization
Explicit Semantic Analysis	DBMS_DATA_MINING —Algorithm Settings: Explicit Semantic Analysis
Exponential Smoothing	DBMS_DATA_MINING —Algorithm Settings: Exponential Smoothing Models
Generalized Linear Model	DBMS_DATA_MINING —Algorithm Settings: Generalized Linear Models
k-Means	DBMS_DATA_MINING —Algorithm Settings: *k*-Means
Multivariate State Estimation Technique - Sequential Probability Ratio Test	DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test
Naive Bayes	Algorithm Settings: Naive Bayes
Neural Network	DBMS_DATA_MINING —Algorithm Settings: Neural Network
Non-Negative Matrix Factorization	DBMS_DATA_MINING —Algorithm Settings: Non-Negative Matrix Factorization
O-Cluster	Algorithm Settings: O-Cluster
Random Forest	DBMS_DATA_MINING — Algorithm Settings: Random Forest

Table 36-11 (Cont.) Algorithm-Specific Model Settings

Algorithm	Description
Singular Value Decomposition	DBMS_DATA_MINING —Algorithm Settings: Singular Value Decomposition
Support Vector Machine	DBMS_DATA_MINING —Algorithm Settings: Support Vector Machine
XGBoost	DBMS_DATA_MINING — Algorithm Settings: XGBoost

> **Note:**
>
> Some XGBoost objectives apply only to classification function models and other objectives apply only to regression function models. If you specify an incompatible `objective` value, an error is raised. In the `DBMS_DATA_MINING.CREATE_MODEL` procedure, if you specify `DBMS_DATA_MINING.CLASSIFICATION` as the function, then the only objective values that you can use are the `binary` and `multi` values. The one exception is `binary: logitraw`, which produces a continuous value and applies only to a regression model. If you specify `DBMS_DATA_MINING.REGRESSION` as the function, then you can specify `binary: logitraw` or any of the `count`, `rank`, `reg`, and `survival` values as the objective.
>
> The values for the XGBoost objective setting are listed in the Settings for Learning Tasks table in DBMS_DATA_MINING — Algorithm Settings: XGBoost.

Example 36-3 Creating a Settings Table and Creating an SVM Classification Model Using CREATE.MODEL procedure

```
CREATE TABLE svmc_sh_sample_settings (
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(4000));

BEGIN
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.svms_kernel_function, dbms_data_mining.svms_linear);
  COMMIT;
END;
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svmc_sh_sample_settings');
END;
```

Example 36-4 Specify Model Settings for a SVM Classification Model Using CREATE_MODEL2 procedure

```
DECLARE
    v_setlist DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlist('PREP_AUTO') := 'ON';
    v_setlist('ALGO_NAME') := 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlist('SVMS_KERNEL_FUNCTION') := 'SVMS_LINEAR';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MODEL',
        MINING_FUNCTION     => 'CLASSIFICATION',
        DATA_QUERY          => 'select * from mining_data_build_v',
        SET_LIST            => v_setlist,
        CASE_ID_COLUMN_NAME => 'CUST_ID,
    TARGET_COLUMN_NAME  => 'AFFINITY_CARD');
END;
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

36.7.1 Specify Costs

Specify a cost matrix table to build a Decision Tree model.

The `CLAS_COST_TABLE_NAME` setting specifies the name of a cost matrix table to be used in building a Decision Tree model. A cost matrix biases a classification model to minimize costly misclassifications. The cost matrix table must have the columns shown in the following table:

Table 36-12 Cost Matrix Table Required Columns

Column Name	Data Type
`actual_target_value`	valid target data type
`predicted_target_value`	valid target data type
`cost`	`NUMBER`

Decision Tree is the only algorithm that supports a cost matrix at build time. However, you can create a cost matrix and associate it with any classification model for scoring.

If you want to use costs for scoring, create a table with the columns shown in Table 36-12, and use the `DBMS_DATA_MINING.ADD_COST_MATRIX` procedure to add the cost matrix table to the model. You can also specify a cost matrix inline when invoking a `PREDICTION` function. Table 35-1 has details for valid target data types.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

36.7.2 Specify Prior Probabilities

Prior probabilities can be used to offset differences in distribution between the build data and the actual population.

The `CLAS_PRIORS_TABLE_NAME` setting specifies the name of a table of prior probabilities to be used in building a Naive Bayes model. The priors table must have the columns shown in the following table.

Table 36-13 Priors Table Required Columns

Column Name	Data Type
`target_value`	valid target data type
`prior_probability`	`NUMBER`

Related Topics

- Target Attribute
  Understand what a **target** means in machine learning and understand the different target data types.
- *Oracle Machine Learning for SQL Concepts*

36.7.3 Specify Class Weights

Specify class weights table settings in logistic regression or Support Vector Machine (SVM) classification to favor higher weighted classes.

The `CLAS_WEIGHTS_TABLE_NAME` setting specifies the name of a table of class weights to be used to bias a logistic regression (Generalized Linear Model classification) or SVM classification model to favor higher weighted classes. The weights table must have the columns shown in the following table.

Table 36-14 Class Weights Table Required Columns

Column Name	Data Type
`target_value`	Valid target data type
`class_weight`	`NUMBER`

Related Topics

- Target Attribute
  Understand what a **target** means in machine learning and understand the different target data types.
- *Oracle Machine Learning for SQL Concepts*

36.7.4 Model Settings in the Data Dictionary

Explains about `ALL/USER/DBA_MINING_MODEL_SETTINGS` in data dictionary view.

Information about Oracle Machine Learning model settings can be obtained from the data dictionary view `ALL/USER/DBA_MINING_MODEL_SETTINGS`. When used with the `ALL` prefix, this view returns information about the settings for the models accessible to the current user. When used with the `USER` prefix, it returns information about the settings for the models in the user's schema. The `DBA` prefix is only available for DBAs.

The columns of `ALL_MINING_MODEL_SETTINGS` are described as follows and explained in the following table.

```
SQL> describe all_mining_model_settings
```

The output is as follows:

```
Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(30)
 MODEL_NAME                                NOT NULL VARCHAR2(30)
 SETTING_NAME                              NOT NULL VARCHAR2(30)
 SETTING_VALUE                                      VARCHAR2(4000)
 SETTING_TYPE                                       VARCHAR2(7)
```

Table 36-15 ALL_MINING_MODEL_SETTINGS

Column	Description
`owner`	Owner of the machine learning model.
`model_name`	Name of the machine learning model.
`setting_name`	Name of the setting.
`setting_value`	Value of the setting.
`setting_type`	`INPUT` if the value is specified by a user. `DEFAULT` if the value is system-generated.

The following query lists the settings for the Support Vector Machine (SVM) classification model SVMC_SH_CLAS_SAMPLE. The `ALGO_NAME`, `CLAS_WEIGHTS_TABLE_NAME`, and `SVMS_KERNEL_FUNCTION` settings are user-specified. These settings have been specified in a settings table for the model. The SVMC_SH_CLAS_SAMPLE model is created by the `oml4sql-classification-svm.sql` example.

Example 36-5 ALL_MINING_MODEL_SETTINGS

```
SQL> COLUMN setting_value FORMAT A35
SQL> SELECT setting_name, setting_value, setting_type
            FROM all_mining_model_settings
            WHERE model_name in 'SVMC_SH_CLAS_SAMPLE';
```

The output is as follows:

```
SETTING_NAME                  SETTING_VALUE
SETTING
------------------------------ -----------------------------------
-------
SVMS_ACTIVE_LEARNING          SVMS_AL_ENABLE
DEFAULT
PREP_AUTO                     OFF
DEFAULT
SVMS_COMPLEXITY_FACTOR        0.244212
DEFAULT
SVMS_KERNEL_FUNCTION          SVMS_LINEAR
INPUT
CLAS_WEIGHTS_TABLE_NAME       svmc_sh_sample_class_wt
INPUT
SVMS_CONV_TOLERANCE           .001
DEFAULT
ALGO_NAME                     ALGO_SUPPORT_VECTOR_MACHINES
INPUT
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

36.7.5 Specify Oracle Machine Learning Model Settings for an R Model

This topic applies only to Oracle on-premises.

The machine learning model settings for an R language model determine the characteristics of the model and are specified in the model settings table.

You can build a machine learning model in the R language by specifying R as the value of the `ALGO_EXTENSIBLE_LANG` setting in the model settings table. You can create a model by combining in the settings table generic settings that do not require an algorithm, such as `ODMS_PARTITION_COLUMNS` and `ODMS_SAMPLING`. You can also specify the following settings, which are exclusive to an R machine learning model.

- ALGO_EXTENSIBLE_LANG
- RALG_BUILD_FUNCTION
- RALG_BUILD_PARAMETER
- RALG_DETAILS_FORMAT
- RALG_DETAILS_FUNCTION
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

36.7.5.1 ALGO_EXTENSIBLE_LANG

Use the `ALGO_EXTENSIBLE_LANG` setting to specify the language for the Oracle Machine Learning for SQL extensible algorithm framework.

Currently, `R` is the only valid value for the `ALGO_EXTENSIBLE_LANG` setting. When you set the value for `ALGO_EXTENSIBLE_LANG` to R, the machine learning models are built using the R language. You can use the following settings in the settings table to specify the characteristics of the R model.

- RALG_BUILD_FUNCTION
- RALG_BUILD_PARAMETER
- RALG_DETAILS_FUNCTION
- RALG_DETAILS_FORMAT
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

36.7.5.2 RALG_BUILD_FUNCTION

Use the `RALG_BUILD_FUNCTION` setting to specify the name of an existing registered R script for building an Oracle Machine Learning for SQL model using the R language.

You must specify both the `RALG_BUILD_FUNCTION` and `ALGO_EXTENSIBLE_LANG` settings in the model settings table. The R script defines an R function that has as the first input argument an R `data.frame` object for training data. The function returns an Oracle Machine Learning model object. The first data argument is mandatory. The `RALG_BUILD_FUNCTION` can accept additional model build parameters.

> **Note:**
>
> The valid inputs for input parameters are numeric and string scalar data types.

Example 36-6 Example of RALG_BUILD_FUNCTION

This example shows how to specify the name of the R script *`MY_LM_BUILD_SCRIPT`* that is used to build the model.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_build_function,'MY_LM_BUILD_SCRIPT');
```

```
End;
/
```

The R script *MY_LM_BUILD_SCRIPT* defines an R function that builds the LM model. You must register the script *MY_LM_BUILD_SCRIPT* in the Oracle Machine Learning for R script repository which uses the existing OML4R security restrictions. You can use the OML4R `sys.rqScriptCreate` procedure to register the script. OML4R requires the `RQADMIN` role to register R scripts.

For example:

```
Begin
sys.rqScriptCreate('MY_LM_BUILD_SCRIPT', 'function(data, formula,
model.frame) {lm(formula = formula, data=data, model =
as.logical(model.frame)}');
End;
/
```

For Clustering and Feature Extraction machine learning function model builds, the R attributes `dm$nclus` and `dm$nfeat` must be set on the return R model to indicate the number of clusters and features respectively.

The R script `MY_KM_BUILD_SCRIPT` defines an R function that builds the *k*-Means model for clustering. The R attribute `dm$nclus` is set with the number of clusters for the returned clustering model.

```
'function(dat) {dat.scaled <- scale(dat)
     set.seed(6543); mod <- list()
     fit <- kmeans(dat.scaled, centers = 3L)
     mod[[1L]] <- fit
     mod[[2L]] <- attr(dat.scaled, "scaled:center")
     mod[[3L]] <- attr(dat.scaled, "scaled:scale")
     attr(mod, "dm$nclus") <- nrow(fit$centers)
     mod}'
```

The R script `MY_PCA_BUILD_SCRIPT` defines an R function that builds the PCA model. The R attribute `dm$nfeat` is set with the number of features for the returned feature extraction model.

```
'function(dat) {
     mod <- prcomp(dat, retx = FALSE)
     attr(mod, "dm$nfeat") <- ncol(mod$rotation)
     mod}'
```

Related Topics

- RALG_BUILD_PARAMETER
  The `RALG_BUILD_FUNCTION` input parameter specifies a list of numeric and string scalar values in SQL `SELECT` query statement format.
- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

36.7.5.2.1 RALG_BUILD_PARAMETER

The RALG_BUILD_FUNCTION input parameter specifies a list of numeric and string scalar values in SQL SELECT query statement format.

Example 36-7   Example of RALG_BUILD_PARAMETER

The RALG_BUILD_FUNCTION input parameters must be a list of numeric and string scalar values. The input parameters are optional.

The syntax of the parameter is:

```
'SELECT value parameter name ...FROM dual'
```

This example shows how to specify a formula for the input argument 'formula' and a numeric value of zero for input argument 'model.frame' using the RALG_BUILD_PARAMETER. These input arguments must match with the function signature of the R script used in the RALG_BUILD_FUNCTION parameter.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_build_parameter, 'select ''AGE ~ .'' as "formula", 0
as "model.frame" from dual');
End;
/
```

Related Topics

- RALG_BUILD_FUNCTION
  Use the RALG_BUILD_FUNCTION setting to specify the name of an existing registered R script for building an Oracle Machine Learning for SQL model using the R language.

36.7.5.3 RALG_DETAILS_FUNCTION

The RALG_DETAILS_FUNCTION specifies the R model metadata that is returned in the R data.frame.

Use the RALG_DETAILS_FUNCTION to specify an existing registered R script that generates model information. The script defines an R function that contains the first input argument for the R model object. The output of the R function must be a data.frame. The columns of the data.frame are defined by the RALG_DETAILS_FORMAT setting, and may contain only numeric or string scalar types.

Example 36-8   Example of RALG_DETAILS_FUNCTION

This example shows how to specify the name of the R script MY_LM_DETAILS_SCRIPT in the model settings table. This script defines the R function that is used to provide the model information.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_details_function, 'MY_LM_DETAILS_SCRIPT');
```

```
End;
/
```

In the Oracle Machine Learning for R script repository, the script *MY_LM_DETAILS_SCRIPT* is registered as:

```
 'function(mod) data.frame(name=names(mod$coefficients),
    coef=mod$coefficients)'
```

Related Topics

- Registered R Scripts
  The RALG_*_FUNCTION settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.
- RALG_DETAILS_FORMAT
  Use the RALG_DETAILS_FORMAT setting to specify the names and column types in the model view.

36.7.5.3.1 RALG_DETAILS_FORMAT

Use the RALG_DETAILS_FORMAT setting to specify the names and column types in the model view.

The value of the setting is a string that contains a SELECT statement to specify a list of numeric and string scalar data types for the name and type of the model view columns.

When the RALG_DETAILS_FORMAT and RALG_DETAILS_FUNCTION settings are both specified, a model view by the name DM$VD *<model_name>* is created along with an R model in the current schema. The first column of the model view is PARTITION_NAME. It has the value NULL for non-partitioned models. The other columns of the model view are defined by RALG_DETAILS_FORMAT setting.

Example 36-9 Example of RALG_DETAILS_FORMAT

This example shows how to specify the name and type of the columns for the generated model view. The model view contains the varchar2 column attr_name and the number column coef_value after the first column partition_name.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_details_format, 'select cast(''a'' as
varchar2(20)) as attr_name, 0 as coef_value from dual');
End;
/
```

Related Topics

- RALG_DETAILS_FUNCTION
  The RALG_DETAILS_FUNCTION specifies the R model metadata that is returned in the R data.frame.

36.7.5.4 RALG_SCORE_FUNCTION

Use the `RALG_SCORE_FUNCTION` setting to specify an existing registered R script for R algorithm machine learning model to use for scoring data.

The specified R script defines an R function. The first input argument defines the model object. The second input argument defines the R `data.frame` that is used for scoring data.

Example 36-10 Example of RALG_SCORE_FUNCTION

This example shows how the R function takes the Linear Model model and scores the data in the `data.frame`. The function argument `object` is the LM model. The argument `newdata` is a `data.frame` containing the data to score.

```
function(object, newdata) {res <- predict.lm(object, newdata = newdata,
se.fit = TRUE); data.frame(fit=res$fit, se=res$se.fit,
df=summary(object)$df[1L])}
```

The output of the R function must be a `data.frame`. Each row represents the prediction for the corresponding scoring data from the input `data.frame`. The columns of the `data.frame` are specific to machine learning functions, such as:

Regression: A single numeric column for the predicted target value, with two optional columns containing the standard error of the model fit, and the degrees of freedom number. The optional columns are needed for the SQL function `PREDICTION_BOUNDS` to work.

Example 36-11 Example of RALG_SCORE_FUNCTION for Regression

This example shows how to specify the name of the R script *`MY_LM_PREDICT_SCRIPT`* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_LM_PREDICT_SCRIPT');
End;
/
```

In the Oracle Machine Learning for R script repository, the script *`MY_LM_PREDICT_SCRIPT`* is registered as:

```
function(object, newdata) {data.frame(pre = predict(object, newdata =
newdata))}
```

Classification: Each column represents the predicted probability of one target class. The column name is the target class name.

Example 36-12 Example of RALG_SCORE_FUNCTION for Classification

This example shows how to specify the name of the R script *`MY_LOGITGLM_PREDICT_SCRIPT`* that is used to score the logit Classification model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
```

```
(dbms_data_mining.ralg_score_function, 'MY_LOGITGLM_PREDICT_SCRIPT');
End;
/
```

In the OML4R script repository, *MY_LOGITGLM_PREDICT_SCRIPT* is registered as follows. It is a logit Classification with two target classes, "0" and "1".

```
'function(object, newdata) {
   pred <- predict(object, newdata = newdata, type="response");
   res <- data.frame(1-pred, pred);
   names(res) <- c("0", "1");
   res}'
```

Clustering: Each column represents the predicted probability of one cluster. The columns are arranged in order of cluster ID. Each cluster is assigned a cluster ID, and they are consecutive values starting from 1. To support `CLUSTER_DISTANCE` in the R model, the output of R score function returns an extra column containing the value of the distance to each cluster in order of cluster ID after the columns for the predicted probability.

Example 36-13 Example of RALG_SCORE_FUNCTION for Clustering

This example shows how to specify the name of the R script *MY_CLUSTER_PREDICT_SCRIPT* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_CLUSTER_PREDICT_SCRIPT');
End;
/
```

In the OML4R script repository, the script *MY_CLUSTER_PREDICT_SCRIPT* is registered as:

```
'function(object, dat){
     mod <- object[[1L]]; ce <- object[[2L]]; sc <- object[[3L]];
     newdata = scale(dat, center = ce, scale = sc);
     centers <- mod$centers;
     ss <- sapply(as.data.frame(t(centers)),
     function(v) rowSums(scale(newdata, center=v, scale=FALSE)^2));
     if (!is.matrix(ss)) ss <- matrix(ss, ncol=length(ss));
     disp <- -1 / (2* mod$tot.withinss/length(mod$cluster));
     distr <- exp(disp*ss);
     prob <- distr / rowSums(distr);
     as.data.frame(cbind(prob, sqrt(ss)))}'
```

Feature Extraction: Each column represents the coefficient value of one feature. The columns are arranged in order of feature ID. Each feature is assigned a feature ID, which are consecutive values starting from 1.

Example 36-14 Example of RALG_SCORE_FUNCTION for Feature Extraction

This example shows how to specify the name of the R script *`MY_FEATURE_EXTRACTION_SCRIPT`* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_FEATURE_EXTRACTION_SCRIPT');
End;
/
```

In the OML4R script repository, the script *`MY_FEATURE_EXTRACTION_SCRIPT`* is registered as:

```
 'function(object, dat) { as.data.frame(predict(object, dat)) }'
```

The function fetches the centers of the features from the R model, and computes the feature coefficient based on the distance of the score data to the corresponding feature center.

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

36.7.5.5 RALG_WEIGHT_FUNCTION

Use the `RALG_WEIGHT_FUNCTION` setting to specify the name of an existing registered R script that computes the weight or contribution for each attribute in scoring. The specified R script is used in the SQL function `PREDICTION_DETAILS` to evaluate attribute contribution.

The specified R script defines an R function containing the first input argument for a model object, and the second input argument of an R `data.frame` for scoring data. When the machine learning function is Classification, Clustering, or Feature Extraction, the target class name, cluster ID, or feature ID is passed by the third input argument to compute the weight for that particular class, cluster, or feature. The script returns a `data.frame` containing the contributing weight for each attribute in a row. Each row corresponds to that input scoring `data.frame`.

Example 36-15 Example of RALG_WEIGHT_FUNCTION

This example specifies the name of the R script *`MY_PREDICT_WEIGHT_SCRIPT`* that computes the weight or contribution of R model attributes in the `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_weight_function, 'MY_PREDICT_WEIGHT_SCRIPT');
End;
/
```

In the Oracle Machine Learning for R script repository, the script *`MY_PREDICT_WEIGHT_SCRIPT`* for Regression is registered as:

```
'function(mod, data) { coef(mod)[-1L]*data }'
```

In the OML4R script repository, the script *MY_PREDICT_WEIGHT_SCRIPT* for logit Classification is registered as:

```
'function(mod, dat, clas) {
   v <- predict(mod, newdata=dat, type = "response");
   v0 <- data.frame(v, 1-v); names(v0) <- c("0", "1");
   res <- data.frame(lapply(seq_along(dat),
   function(x, dat) {
   if(is.numeric(dat[[x]])) dat[,x] <- as.numeric(0)
   else dat[,x] <- as.factor(NA);
   vv <- predict(mod, newdata = dat, type = "response");
   vv = data.frame(vv, 1-vv); names(vv) <- c("0", "1");
   v0[[clas]] / vv[[clas]]}, dat = dat));
   names(res) <- names(dat);
   res}'
```

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

36.7.5.6 Registered R Scripts

The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

You can register the R scripts using the OML4R SQL procedure `sys.rqScriptCreate`. To register a scripts, you must have the `RQADMIN` role.

The `RALG_*_FUNCTION` settings include the following functions:

- RALG_BUILD_FUNCTION
- RALG_DETAILS_FUNCTION
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

> **Note:**
>
> The R scripts must exist in the OML4R script repository for an R model to function.

After an R model is built, the name of the specified R script become a model setting. These R script must exist in the OML4R script repository for an R model to remain functional.

You can manage the R memory that is used to build, score, and view the R models through OML4R as well.

36.7.5.7 R Model Demonstration Scripts

You can access R model demonstration scripts under `rdbms/demo`

```
dmraidemo.sql   dmrglmdemo.sql dmrpcademo.sql
dmrardemo.sql   dmrkmdemo.sql  dmrrfdemo.sql
dmrdtdemo.sql   dmrnndemo.sql
```

36.8 Model Detail Views

Model detail views are algorithm-specific. Viewing the model detail views will provide you with additional information about the model you created. The names of model detail views begin with DM$. Some model views, such as Global Name-Value Pairs view (`DM$VG`*model_name*), Computed Settings view (`DM$VS`*model_name*), Model Build Alerts view (`DM$VW`*model_name*), and Normalization and Missing Value Handling view (`DM$VN`*model_name*), are shared by all algorithms and are documented separately. Aside from that, classification, clustering, and regression algorithms share some common views. The columns returned by these views may differ between algorithms.

The following are the model views, grouped by model function:

Association:

- Model Detail Views for Association Rules
- Model Detail View for Frequent Itemsets
- Model Detail Views for Transactional Itemsets
- Model Detail View for Transactional Rule

Classification, Regression, and Anomaly Detection:

- Model Detail Views for Classification Algorithms
- Model Detail Views for CUR Matrix Decomposition
- Model Detail Views for Decision Tree
- Model Detail Views for Generalized Linear Model
- Model Detail View for Multivariate State Estimation Technique - Sequential Probability Ratio Test
- Model Detail Views for Naive Bayes
- Model Detail Views for Neural Network
- Model Detail Views for Random Forest
- Model Detail View for Support Vector Machine
- Model Detail Views for XGBoost

Clustering:

- Model Detail Views for Clustering Algorithms
- Model Detail Views for Expectation Maximization
- Model Detail Views for *k*-Means

- Model Detail Views for O-Cluster

Feature Extraction:

- Model Detail Views for Explicit Semantic Analysis
- Model Detail Views for Non-Negative Matrix Factorization
- Model Detail Views for Singular Value Decomposition

Feature Selection:

- Model Detail Views for Minimum Description Length

Data Preparation and Other:

- Model Detail Views for Binning
- Model Detail Views for Global Information
- Model Detail Views for Normalization and Missing Value Handling

Time Series:

Model Detail Views for Exponential Smoothing

ONNX Models:

Model Detail Views for ONNX Models

36.8.1 Model Detail Views for Association Rules

The model detail view `DM$VR`*model_name* contains the generated rules for association models.

These are the available model views for Association Rules:

Model Views	Description
`DM$VA`*model_name*	Association Rules For Transactional Data
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VI`*model_name*:	Association Rule Itemsets
`DM$VR`*model_name*	Association Rules
`DM$VS`*model_name*	Computed Settings
`DM$VT`*model_name*	Association Rule Itemsets For Transactional Data
`DM$VW`*model_name*	Model Build Alerts

Depending on the settings of the model, this rule view (`DM$VR`*model_name*) different sets of columns. Settings `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` determine how each item is defined. If `ODMS_ITEM_ID_COLUMN_NAME` is set, the input format is called transactional input, otherwise, the input format is called 2-Dimensional input. With transactional input, if setting `ODMS_ITEM_VALUE_COLUMN_NAME` is not set, each item is defined by `ITEM_NAME`, otherwise, each item is defined by `ITEM_NAME` and `ITEM_VALUE`. With 2-Dimensional input, each item is defined by `ITEM_NAME`, `ITEM_SUBNAME` and `ITEM_VALUE`. Setting `ASSO_AGGREGATES` specifies the columns to aggregate, which is displayed in the view.

> **Note:**
>
> Setting `ASSO_AGGREGATES` is not allowed for 2-dimensional input.

The following shows the views with different settings.

Transactional Input Without ASSO_AGGREGATES Setting

When you sett `ITEM_NAME` (`ODMS_ITEM_ID_COLUMN_NAME`) and do not set `ITEM_VALUE` (`ODMS_ITEM_VALUE_COLUMN_NAME`), the view contains the following. The consequent item is defined with only the name field. If you also set `ITEM_VALUE`, the view has the additional column `CONSEQUENT_VALUE` that specifies the value field.

```
Name                                     Type
 ----------------------------------------- ----------------------------
 PARTITION_NAME                           VARCHAR2(128)
 RULE_ID                                  NUMBER
 RULE_SUPPORT                             NUMBER
 RULE_CONFIDENCE                          NUMBER
 RULE_LIFT                                NUMBER
 RULE_REVCONFIDENCE                       NUMBER
 ANTECEDENT_SUPPORT                       NUMBER
 NUMBER_OF_ITEMS                          NUMBER
 CONSEQUENT_SUPPORT                       NUMBER
 CONSEQUENT_NAME                          VARCHAR2(4000)
 ANTECEDENT                               SYS.XMLTYPE
```

Table 36-16 Rule View Columns for Transactional Inputs

Column Name	Description
`PARTITION_NAME`	A partition in a partitioned model to retrieve details.
`RULE_ID`	The identifier of the rule.
`RULE_SUPPORT`	The number of transactions that satisfy the rule.
`RULE_CONFIDENCE`	The likelihood of a transaction satisfying the rule.
`RULE_LIFT`	The degree of improvement in the prediction over random chance when the rule is satisfied.
`RULE_REVCONFIDENCE`	The number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs.
`ANTECEDENT_SUPPORT`	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions.
`NUMBER_OF_ITEMS`	The total number of attributes referenced in the antecedent and consequent of the rule.
`CONSEQUENT_SUPPORT`	The ratio of the number of transactions that satisfy the consequent to the total number of transactions.
`CONSEQUENT_NAME`	The name of the consequent.
`CONSEQUENT_VALUE`	The value of the consequent. This column is present when `Item_value` (`ODMS_ITEM_VALUE_COLUMN_NAME`) is set with `TYPE` as numerical or categorical.

Table 36-16 (Cont.) Rule View Columns for Transactional Inputs

Column Name	Description
ANTECEDENT	The antecedent is described as an itemset. At the itemset level, it specifies the number of aggregates, and if not zero, the names of the columns to be aggregated (as well as the mapping to ASSO_AGG*). The itemset contains >= 1 items. • When ODMS_ITEM_VALUE_COLUMN_NAME is not set, each item is defined by item_name. As an example, if the antecedent contains one item B, then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>B</item_name></item></itemset> As another example, if the antecedent contains two items, A and C, then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>A</item_name></item><item><item_name>C</item_name></item></itemset> • When setting ODMS_ITEM_VALUE_COLUMN_NAME is set, each item is defined by item_name and item_value. As an example, if the antecedent contains two items, (name A, value 1) and (name C, value 1), then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>A</item_name><item_value>1</item_value></item><item><item_name>C</item_name><item_value>1</item_value></item></itemset>

Transactional Input With ASSO_AGGREGATES Setting

Similar to the view without an aggregates setting, there are three cases:

- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is not set.
- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as numerical, the view has a CONSEQUENT_VALUE column.
- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as categorical, the view has a CONSEQUENT_VALUE column.

For the example that produces the following rules, see “Example: Calculating Aggregates” in *Oracle Machine Learning for SQL Concepts*.

The view reports two sets of aggregates results:

1. ANT_RULE_PROFIT refers to the total profit for the antecedent itemset with respect to the rule, the profit for each individual item of the antecedent itemset is shown in the ANTECEDENT (XMLtype) column, CON_RULE_PROFIT refers to the total profit for the consequent item with respect to the rule.

In the example, for rule (A, B) => C, the rule itemset (A, B, C) occurs in the transactions of customer 1 and customer 3. The `ANT_RULE_PROFIT` is $21.20, The `ANTECEDENT` is shown as follow, which tells that item A has profit 5.00 + 3.00 = $8.00 and item B has profit 3.20 + 10.00 = $13.20, which sum up to `ANT_RULE_PROFIT`.

```
<itemset NUMAGGR="1" ASSO_AGG0="profit"><item><item_name>A</
item_name><ASSO_AGG0>8.0E+000</ASSO_AGG0></item><item><item_name>B</
item_name><ASSO_AGG0>1.32E+001</ASSO_AGG0></item></itemset>
The CON_RULE_PROFIT is 12.00 + 14.00 = $26.00
```

2. `ANT_PROFIT` refers to the total profit for the antecedent itemset, while `CON_PROFIT` refers to the total profit for the consequent item. The difference between `CON_PROFIT` and `CON_RULE_PROFIT` (the same applies to `ANT_PROFIT` and `ANT_RULE_PROFIT`) is that `CON_PROFIT` counts all profit for the consequent item across all transactions where the consequent occurs, while `CON_RULE_PROFIT` only counts across transactions where the rule itemset occurs.

   For example, item C occurs in transactions for customer 1, 2 and 3, `CON_PROFIT` is 12.00 + 4.20 + 14.00 = $30.20, while `CON_RULE_PROFIT` only counts transactions for customer 1 and 3 where the rule itemset (A, B, C) occurs.

   Similarly, `ANT_PROFIT` counts all transactions where itemset (A, B) occurs, while `ANT_RULE_PROFIT` counts only transactions where the rule itemset (A, B, C) occurs. In this example, by coincidence, both count transactions for customer 1 and 3, and have the same value.

Example 36-16 Examples

The following example shows the view when setting `ASSO_AGGREGATES` specifies column profit and column sales to be aggregated. In this example, `ITEM_VALUE` column is not specified.

```
Name                                      Type
 ----------------------------------------- ----------------------------
 PARTITION_NAME                            VARCHAR2(128)
 RULE_ID                                   NUMBER
 RULE_SUPPORT                              NUMBER
 RULE_CONFIDENCE                           NUMBER
 RULE_LIFT                                 NUMBER
 RULE_REVCONFIDENCE                        NUMBER
 ANTECEDENT_SUPPORT                        NUMBER
 NUMBER_OF_ITEMS                           NUMBER
 CONSEQUENT_SUPPORT                        NUMBER
 CONSEQUENT_NAME                           VARCHAR2(4000)
 ANTECEDENT                                SYS.XMLTYPE
 ANT_RULE_PROFIT                           BINARY_DOUBLE
 CON_RULE_PROFIT                           BINARY_DOUBLE
 ANT_PROFIT                                BINARY_DOUBLE
 CON_PROFIT                                BINARY_DOUBLE
 ANT_RULE_SALES                            BINARY_DOUBLE
 CON_RULE_SALES                            BINARY_DOUBLE
 ANT_SALES                                 BINARY_DOUBLE
 CON_SALES                                 BINARY_DOUBLE
```

The rule view has a `CONSEQUENT_VALUE` column when `ODMS_ITEM_ID_COLUMN_NAME` is set and `Item_value` (`ODMS_ITEM_VALUE_COLUMN_NAME`) is set with `TYPE` as numerical or categorical.

2-Dimensional Inputs

In Oracle Machine Learning for SQL, association models can be built using either transactional or two-dimensional data formats. For two-dimensional input, each item is defined by three fields: `NAME`, `VALUE` and `SUBNAME`. The `NAME` field is the name of the column. The `VALUE` field is the content of the column. The `SUBNAME` field is used when the input data table contains a nested table. In that case, `SUBNAME` is the name of the nested table's column. See, Example: Creating a Nested Column for Market Basket Analysis. In this example, there is a nested column. The `CONSEQUENT_SUBNAME` is the `ATTRIBUTE_NAME` part of the nested column. That is, `'O/S Documentation Set - English'` and `CONSEQUENT_VALUE` is the value part of the nested column, which is, 1.

The view uses three columns for the consequent. The rule view has the following columns:

```
Name                                      Type
 ----------------------- ---------------------
 PARTITION_NAME                            VARCHAR2(128)
 RULE_ID                                   NUMBER
 RULE_SUPPORT                              NUMBER
 RULE_CONFIDENCE                           NUMBER
 RULE_LIFT                                 NUMBER
 RULE_REVCONFIDENCE                        NUMBER
 ANTECEDENT_SUPPORT                        NUMBER
 NUMBER_OF_ITEMS                           NUMBER
 CONSEQUENT_SUPPORT                        NUMBER
 CONSEQUENT_NAME                           VARCHAR2(4000)
 CONSEQUENT_SUBNAME                        VARCHAR2(4000)
 CONSEQUENT_VALUE                          VARCHAR2(4000)
 ANTECEDENT                                SYS.XMLTYPE
```

Note:

All of the types for three columns for the consequent are `VARCHAR2`. `ASSO_AGGREGATES` is not applicable for 2-Dimensional input format.

The following table displays rule view columns for 2-Dimensional input with the descriptions of only the fields that are specific to 2-D inputs.

Table 36-17 Rule View for 2-Dimensional Input

Column Name	Description
`CONSEQUENT_SUBNAME`	For two-dimensional inputs, `CONSEQUENT_SUBNAME` is used for nested column in the input data table.
`CONSEQUENT_VALUE`	The value of the consequent when setting `Item_value` is set with `TYPE` as numerical or categorical.

Table 36-17 (Cont.) Rule View for 2-Dimensional Input

Column Name	Description
`ANTECEDENT`	The antecedent is described as an itemset. The itemset contains `>= 1` items. Each item is defined using `ITEM_NAME`, `ITEM_SUBNAME`, and `ITEM_VALUE`: As an example, assuming that this is not a nested table input, and the antecedent contains one item: (name `ADDR`, value `MA`). The antecedent (XMLtype) is as follows: `<itemset NUMAGGR="0"><item><item_name>ADDR</item_name><item_subname></item_subname><item_value>MA</item_value></item></itemset>` For 2-Dimensional input with nested table, the subname field is filled.

Global Name-Value Pairs View for Association Rules

Global Name-Value Pairs View produces a single column for an association model. The following table describes the columns returned for association model.

Table 36-18 Global Name-Value Pairs View for an Association Model

Name	Description
`ITEMSET_COUNT`	The number of itemsets generated.
`MAX_SUPPORT`	The maximum support.
`NUM_ROWS`	The total number of rows used in the build.
`RULE_COUNT`	The number of association rules in the model generated.
`TRANSACTION_COUNT`	The number of the transactions in the input data.

36.8.2 Model Detail View for Frequent Itemsets

The model detail view `DM$VI`*model_name* contains information about frequent itemsets.

The Association Rule Itemsets view (`DM$VI`*model_name*) has the following columns:

```
Name                Type
-------------       ------------------
PARTITION_NAME      VARCHAR2 (128)
ITEMSET_ID          NUMBER
SUPPORT             NUMBER
NUMBER_OF_ITEMS     NUMBER
 ITEMSET             SYS.XMLTYPE
```

Table 36-19 Association Rule Itemsets View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
ITEMSET_ID	Itemset identifier
SUPPORT	Support of the itemset
NUMBER_OF_ITEMS	Number of items in the itemset
ITEMSET	Frequent itemset The structure of the SYS.XMLTYPE column itemset is the same as the corresponding Antecedent column of the rule view.

36.8.3 Model Detail Views for Transactional Itemsets

The model detail view `DM$VT`*model_name* contains information about the transactional itemsets.

For the very common case of transactional data without aggregates, the Association Rule Itemsets For Transactional Data view (`DM$VT`*model_name*) provides the itemsets information in transactional format. This view can help improve performance for some queries as compared to the view with the XML column. The transactional itemsets view has the following columns:

```
Name                Type
-----------------   -----------------
PARTITION_NAME      VARCHAR2(128)
ITEMSET_ID          NUMBER
ITEM_ID             NUMBER
SUPPORT             NUMBER
NUMBER_OF_ITEMS     NUMBER
ITEM_NAME           VARCHAR2(4000)
```

Table 36-20 Association Rule Itemsets For Transactional Data View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
ITEMSET_ID	Itemset identifier
ITEM_ID	Item identifier
SUPPORT	Support of the itemset
NUMBER_OF_ITEMS	Number of items in the itemset
ITEM_NAME	The name of the item

36.8.4 Model Detail View for Transactional Rule

The model detail view `DM$VA`*model_name* contains information about transactional rules and transactional itemsets.

Transactional data without aggregates also has an Association Rules For Transactional Data view (`DM$VA`*model_name*). This view can improve performance for some queries as compared to the view with the XML column. The transactional rule view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
RULE_ID                                   NUMBER
ANTECEDENT_PREDICATE                      VARCHAR2(4000)
CONSEQUENT_PREDICATE                      VARCHAR2(4000)
RULE_SUPPORT                              NUMBER
RULE_CONFIDENCE                           NUMBER
RULE_LIFT                                 NUMBER
RULE_REVCONFIDENCE                        NUMBER
RULE_ITEMSET_ID                           NUMBER
ANTECEDENT_SUPPORT                        NUMBER
CONSEQUENT_SUPPORT                        NUMBER
NUMBER_OF_ITEMS                           NUMBER
```

Table 36-21 Association Rules For Transactional Data View

Column Name	Description
`PARTITION_NAME`	A partition in a partitioned model
`RULE_ID`	Rule identifier
`ANTECEDENT_PREDICATE`	Name of the Antecedent item.
`CONSEQUENT_PREDICATE`	Name of the Consequent item
`RULE_SUPPORT`	Support of the rule
`RULE_CONFIDENCE`	The likelihood a transaction satisfies the rule when it contains the Antecedent.
`RULE_LIFT`	The degree of improvement in the prediction over random chance when the rule is satisfied
`RULE_REVCONFIDENCE`	The number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs
`RULE_ITEMSET_ID`	Itemset identifier
`ANTECEDENT_SUPPORT`	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions
`CONSEQUENT_SUPPORT`	The ratio of the number of transactions that satisfy the consequent to the total number of transactions
`NUMBER_OF_ITEMS`	Number of items in the rule

36.8.5 Model Detail Views for Classification Algorithms

Model detail views for classification algorithms are the target map view and scoring cost view, which are applicable to all classification algorithms.

These are the available model views for Classification algorithm:

Model Views	Description
DM$VA*model_name*	Variable Importance
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*:	Model Build Alerts

The Classification Targets view (DM$VT*model_name*) describes the target distribution for classification models. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_VALUE                              NUMBER/VARCHAR2
TARGET_COUNT                              NUMBER
TARGET_WEIGHT                             NUMBER
```

Table 36-22 Classification Targets View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
TARGET_VALUE	Target value, numerical or categorical
TARGET_COUNT	Number of rows for a given TARGET_VALUE
TARGET_WEIGHT	Weight for a given TARGET_VALUE

The Scoring Cost Matrix view (DM$VC*model_name*) describes the scoring cost matrix for classification models. The view has the following columns:

```
Name                                      Type
-----------------------------------------
--------------------------------
PARTITION_NAME                            VARCHAR2(128)
ACTUAL_TARGET_VALUE                       NUMBER/VARCHAR2
PREDICTED_TARGET_VALUE                    NUMBER/VARCHAR2
COST                                      NUMBER
```

Table 36-23 Scoring Cost Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ACTUAL_TARGET_VALUE	A valid target value
PREDICTED_TARGET_VALUE	Predicted target value
COST	Associated cost for the actual and predicted target value pair

36.8.6 Model Detail Views for Decision Tree

The model detail views specific to Decision Tree are the hierarchy view, node statistics view, node description view, and the cost matrix view.

These are the model views available for Decision Tree:

Model Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	Decision Tree Statistics
DM$VM*model_name*	Decision Tree Build Cost Matrix
DM$VO*model_name*	Decision Tree Nodes
DM$VP*model_name*	Decision Tree Hierarchy
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The Decision Tree Hierarchy view (DM$VP*model_name*) describes the decision tree hierarchy and the split information for each level in the decision tree. The view has the following columns:

```
Name                                  Type
---------------------------------- ---------------------------
PARTITION_NAME                     VARCHAR2(128)
PARENT                             NUMBER
SPLIT_TYPE                         VARCHAR2
NODE                               NUMBER
ATTRIBUTE_NAME                     VARCHAR2(128)
ATTRIBUTE_SUBNAME                  VARCHAR2(4000)
OPERATOR                           VARCHAR2
VALUE                              SYS.XMLTYPE
```

Table 36-24 Decision Tree Hierarchy View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
PARENT	Node ID of the parent

Table 36-24 (Cont.) Decision Tree Hierarchy View

Column Name	Description
SPLIT_TYPE	The main or surrogate split
NODE	The node ID
ATTRIBUTE_NAME	The attribute used as the splitting criterion at the parent node to produce this node.
ATTRIBUTE_SUBNAME	Split attribute subname. The value is null for non-nested columns.
OPERATOR	Split operator
VALUE	Value used as the splitting criterion. This is an XML element described using the `<Element>` tag. For example, `<Element>Windy</Element><Element>Hot</Element>`.

The Decision Tree Statistics view (`DM$VI`*model_name*) describes the statistics associated with individual tree nodes. The statistics include a target histogram for the data in the node. The view has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
NODE                                  NUMBER
NODE_SUPPORT                          NUMBER
PREDICTED_TARGET_VALUE                NUMBER/VARCHAR2
TARGET_VALUE                          NUMBER/VARCHAR2
TARGET_SUPPORT                        NUMBER
```

Table 36-25 Decision Tree Statistics View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
NODE	The node ID
NODE_SUPPORT	Number of records in the training set that belong to the node
PREDICTED_TARGET_VALUE	Predicted Target value
TARGET_VALUE	A target value seen in the training data
TARGET_SUPPORT	The number of records that belong to the node and have the value specified in the `TARGET_VALUE` column

The Decision Tree Nodes (`DM$VO`*model_name*) view describes higher level node. The `DM$VO`*model_name* has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
NODE                                  NUMBER
```

```
NODE_SUPPORT                            NUMBER
PREDICTED_TARGET_VALUE                  NUMBER/VARCHAR2
PARENT                                  NUMBER
ATTRIBUTE_NAME                          VARCHAR2(128)
ATTRIBUTE_SUBNAME                       VARCHAR2(4000)
OPERATOR                                VARCHAR2
VALUE                                   SYS.XMLTYPE
```

Table 36-26 Decision Tree Nodes View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
NODE	The node ID
NODE_SUPPORT	Number of records in the training set that belong to the node
PREDICTED_TARGET_VALUE	Predicted Target value
PARENT	The ID of the parent
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
OPERATOR	Attribute predicate operator - a conditional operator taking the following values: *IN*, = , <>, < , >, <=, and >=
VALUE	Value used as the description criterion. This is an XML element described using the `<Element>` tag. For example, `<Element>Windy</Element><Element>Hot</Element>`.

The Decision Tree Build Cost Matrix view (`DM$VM`*model_name*) describes the cost matrix used by the Decision Tree build. The `DM$VM`*model_name* view has the following columns:

```
Name                                Type
----------------------------------------- --------------------------------
PARTITION_NAME                           VARCHAR2(128)
ACTUAL_TARGET_VALUE                      NUMBER/VARCHAR2
PREDICTED_TARGET_VALUE                   NUMBER/VARCHAR2
COST                                     NUMBER
```

Table 36-27 Decision Tree Build Cost Matrix View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
ACTUAL_TARGET_VALUE	Valid target value
PREDICTED_TARGET_VALUE	Predicted Target value
COST	Associated cost for the actual and predicted target value pair

The following table describes the Global Name-Value Pairs view (`DM$VG`*model_name*) columns specific to a Decision Tree model.

Table 36-28   Global Name-Value Pairs View

Name	Description
NUM_ROWS	The total number of rows used in the build

36.8.7 Model Detail Views for Generalized Linear Model

Model detail views specific to Generalized Linear Model (GLM) such as details and row diagnostics for linear and logistic regression models are discussed.

The following model views are available for GLM:

Model Views	Description
DM$VA*model_name*	GLM Regression Row Diagnostics
DM$VD*model_name*	GLM Regression Attribute Diagnostics
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The GLM Regression Attribute Diagnostics view (DM$VD*model_name*) describes the final model information for both linear regression models and logistic regression models.

For linear regression, the view DM$VD*model_name* has the following columns:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
ATTRIBUTE_VALUE                       VARCHAR2(4000)
FEATURE_EXPRESSION                    VARCHAR2(4000)
COEFFICIENT                           BINARY_DOUBLE
STD_ERROR                             BINARY_DOUBLE
TEST_STATISTIC                        BINARY_DOUBLE
P_VALUE                               BINARY_DOUBLE
VIF                                   BINARY_DOUBLE
STD_COEFFICIENT                       BINARY_DOUBLE
LOWER_COEFF_LIMIT                     BINARY_DOUBLE
UPPER_COEFF_LIMIT                     BINARY_DOUBLE
```

For logistic regression, the view DM$VD*model_name* has the following columns:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
TARGET_VALUE                          NUMBER/VARCHAR2
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
```

```
ATTRIBUTE_VALUE                    VARCHAR2(4000)
FEATURE_EXPRESSION                 VARCHAR2(4000)
COEFFICIENT                        BINARY_DOUBLE
STD_ERROR                          BINARY_DOUBLE
TEST_STATISTIC                     BINARY_DOUBLE
P_VALUE                            BINARY_DOUBLE
STD_COEFFICIENT                    BINARY_DOUBLE
LOWER_COEFF_LIMIT                  BINARY_DOUBLE
UPPER_COEFF_LIMIT                  BINARY_DOUBLE
EXP_COEFFICIENT                    BINARY_DOUBLE
EXP_LOWER_COEFF_LIMIT              BINARY_DOUBLE
EXP_UPPER_COEFF_LIMIT              BINARY_DOUBLE
```

Table 36-29 Model View for Linear and Logistic Regression Models

Column Name	Description
PARTITION_NAME	The name of a feature in the model
TARGET_VALUE	Valid target value
ATTRIBUTE_NAME	The attribute name when there is no subname, or first part of the attribute name when there is a subname. ATTRIBUTE_NAME is the name of a column in the source table or view. If the column is a non-nested, numeric column, then ATTRIBUTE_NAME is the name of the machine learning attribute. For the intercept, ATTRIBUTE_NAME is null. Intercepts are equivalent to the bias term in SVM models.
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns. When the nested column is numeric, the machine learning attribute is identified by the combination ATTRIBUTE_NAME - ATTRIBUTE_SUBNAME. If the column is not nested, ATTRIBUTE_SUBNAME is null. If the attribute is an intercept, both the ATTRIBUTE_NAME and the ATTRIBUTE_SUBNAME are null.
ATTRIBUTE_VALUE	A unique value that can be assumed by a categorical column or nested categorical column. For categorical columns, a machine learning attribute is identified by a unique ATTRIBUTE_NAME.ATTRIBUTE_VALUE pair. For nested categorical columns, a machine learning attribute is identified by the combination: ATTRIBUTE_NAME.ATTRIBUTE_SUBNAME.ATTRIBUTE_VALUE. For numerical attributes, ATTRIBUTE_VALUE is null.

Table 36-29 (Cont.) Model View for Linear and Logistic Regression Models

Column Name	Description
FEATURE_EXPRESSION	The feature name constructed by the algorithm when feature selection is enabled. If feature selection is not enabled, the feature name is the fully-qualified attribute name (*attribute_name.attribute_subname* if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form: *fully-qualified_attribute_name.attribute_value* When feature generation is enabled, a term in the model can be a single machine learning attribute or the product of up to 3 machine learning attributes. Component machine learning attributes can be repeated within a single term. If feature generation is not enabled or, if feature generation is enabled, but no multiple component terms are discovered by the CREATE model process, then FEATURE_EXPRESSION is null. **Note:** In 12*c* Release 2, the algorithm does not subtract the mean from numerical components.
COEFFICIENT	The estimated coefficient.
STD_ERROR	Standard error of the coefficient estimate.
TEST_STATISTIC	For linear regression, the t-value of the coefficient estimate. For logistic regression, the Wald chi-square value of the coefficient estimate.
P_VALUE	Probability of the TEST_STATISTIC under the (NULL) hypothesis that the term in the model is not statistically significant. A low probability indicates that the term is significant, while a high probability indicates that the term can be better discarded. Used to analyze the significance of specific attributes in the model.
VIF	Variance Inflation Factor. The value is zero for the intercept. For logistic regression, VIF is null.
STD_COEFFICIENT	Standardized estimate of the coefficient.
LOWER_COEFF_LIMIT	Lower confidence bound of the coefficient.
UPPER_COEFF_LIMIT	Upper confidence bound of the coefficient.
EXP_COEFFICIENT	Exponentiated coefficient for logistic regression. For linear regression, EXP_COEFFICIENT is null.
EXP_LOWER_COEFF_LIMIT	Exponentiated coefficient for lower confidence bound of the coefficient for logistic regression. For linear regression, EXP_LOWER_COEFF_LIMIT is null.
EXP_UPPER_COEFF_LIMIT	Exponentiated coefficient for upper confidence bound of the coefficient for logistic regression. For linear regression, EXP_UPPER_COEFF_LIMIT is null.

The GLM Regression Row Diagnostics view `DM$VA`*model_name* describes row level information for both linear regression models and logistic regression models. For linear regression, the view `DM$VA`*model_name* has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
PARTITION_NAME                    VARCHAR2(128)
CASE_ID                           NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                  TIMESTAMP WITH TIME ZONE,
                                  TIMESTAMP WITH LOCAL TIME ZONE
TARGET_VALUE                      BINARY_DOUBLE
PREDICTED_TARGET_VALUE            BINARY_DOUBLE
Hat                               BINARY_DOUBLE
RESIDUAL                          BINARY_DOUBLE
STD_ERR_RESIDUAL                  BINARY_DOUBLE
STUDENTIZED_RESIDUAL              BINARY_DOUBLE
PRED_RES                          BINARY_DOUBLE
COOKS_D                           BINARY_DOUBLE
```

Table 36-30 GLM Regression Row Diagnostics View for Linear Regression

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`CASE_ID`	Name of the case identifier
`TARGET_VALUE`	The actual target value as taken from the input row
`PREDICTED_TARGET_VALUE`	The model predicted target value for the row
`HAT`	The diagonal element of the n*n (n=number of rows) that the Hat matrix identifies with a specific input row. The model predictions for the input data are the product of the Hat matrix and vector of input target values. The diagonal elements (Hat values) represent the influence of the i^{th} row on the i^{th} fitted value. Large Hat values are indicators that the i^{th} row is a point of high leverage, a potential outlier.
`RESIDUAL`	The difference between the predicted and actual target value for a specific input row.
`STD_ERR_RESIDUAL`	The standard error residual, sometimes called the Studentized residual, re-scales the residual to have constant variance across all input rows in an effort to make the input row residuals comparable. The process multiplies the residual by square root of the row weight divided by the product of the model mean square error and 1 minus the Hat value.
`STUDENTIZED_RESIDUAL`	Studentized deletion residual adjusts the standard error residual for the influence of the current row.
`PRED_RES`	The predictive residual is the weighted square of the deletion residuals, computed as the row weight multiplied by the square of the residual divided by 1 minus the Hat value.
`COOKS_D`	Cook's distance is a measure of the combined impact of the i^{th} case on all of the estimated regression coefficients.

For logistic regression, the view `DM$VA`*model_name* has the following columns:

```
Name                               Type
---------------------------------- ----------------------------
PARTITION_NAME                     VARCHAR2(128)
CASE_ID                            NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                   TIMESTAMP WITH TIME ZONE,
                                   TIMESTAMP WITH LOCAL TIME ZONE
TARGET_VALUE                       NUMBER/VARCHAR2
TARGET_VALUE_PROB                  BINARY_DOUBLE
Hat                                BINARY_DOUBLE
WORKING_RESIDUAL                   BINARY_DOUBLE
PEARSON_RESIDUAL                   BINARY_DOUBLE
DEVIANCE_RESIDUAL                  BINARY_DOUBLE
C                                  BINARY_DOUBLE
CBAR                                BINARY_DOUBLE
DIFDEV                             BINARY_DOUBLE
DIFCHISQ                           BINARY_DOUBLE
```

Table 36-31 GLM Regression Row Diagnostics View for Logistic Regression

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`CASE_ID`	Name of the case identifier
`TARGET_VALUE`	The actual target value as taken from the input row
`TARGET_VALUE_PROB`	Model estimate of the probability of the predicted target value.
`Hat`	The Hat value concept from linear regression is extended to logistic regression by multiplying the linear regression Hat value by the variance function for logistic regression, the predicted probability multiplied by 1 minus the predicted probability.
`WORKING_RESIDUAL`	The working residual is the residual of the working response. The working response is the response on the linearized scale. For logistic regression it has the form: the i^{th} row residual divided by the variance of the i^{th} row prediction. The variance of the prediction is the predicted probability multiplied by 1 minus the predicted probability. `WORKING_RESIDUAL` is the difference between the working response and the linear predictor at convergence.
`PEARSON_RESIDUAL`	The Pearson residual is a re-scaled version of the working residual, accounting for the weight. For logistic regression, the Pearson residual multiplies the residual by a factor that is computed as square root of the weight divided by the variance of the predicted probability for the i^{th} row. `RESIDUAL` is 1 minus the predicted probability of the actual target value for the row.
`DEVIANCE_RESIDUAL`	The `DEVIANCE_RESIDUAL` is the contribution to the model deviance of the i^{th} observation. For logistic regression it has the form the square root of 2 times the `log(1 + e^eta) - eta` for the non-reference class and -square root of 2 time the `log (1 + eta)` for the reference class, where `eta` is the linear prediction (the prediction as if the model were a linear regression).

Table 36-31 (Cont.) GLM Regression Row Diagnostics View for Logistic Regression

Column Name	Description
C	Measures the overall change in the fitted logits due to the deletion of the i^{th} observation for all points including the one deleted (the i^{th} point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by the square of 1 minus the Hat value. Confidence interval displacement diagnostics that provides scalar measure of the influence of individual observations.
CBAR	C and CBAR are extensions of Cooks' distance for logistic regression. CBAR measures the overall change in the fitted logits due to the deletion of the i^{th} observation for all points excluding the one deleted (the i^{th} point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by (1 minus the Hat value) Confidence interval displacement diagnostic which measures the influence of deleting an individual observation.
DIFDEV	A statistic that measures the change in deviance that occurs when an observation is deleted from the input. It is computed as the square of the deviance residual plus CBAR.
DIFCHISQ	A statistic that measures the change in the Pearson chi-square statistic that occurs when an observation is deleted from the input. It is computed as CBAR divided by the Hat value.

Global Details for GLM: Linear Regression

The following table describes Global Name-Value Pairs (DM$VG) for a linear regression model.

Table 36-32 Global Details for Linear Regression

Name	Description
ADJUSTED_R_SQUARE	Adjusted R-Square
AIC	Akaike's information criterion
COEFF_VAR	Coefficient of variation
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
CORRECTED_TOTAL_DF	Corrected total degrees of freedom
CORRECTED_TOT_SS	Corrected total sum of squares
DEPENDENT_MEAN	Dependent mean
ERROR_DF	Error degrees of freedom
ERROR_MEAN_SQUARE	Error mean square
ERROR_SUM_SQUARES	Error sum of squares
F_VALUE	Model *F* value statistic
GMSEP	Estimated mean square error of the prediction, assuming multivariate normality

Table 36-32 (Cont.) Global Details for Linear Regression

Name	Description
HOCKING_SP	Hocking Sp statistic
ITERATIONS	Tracks the number of SGD iterations. Applicable only when the solver is SGD.
J_P	JP statistic (the final prediction error)
MODEL_DF	Model degrees of freedom
MODEL_F_P_VALUE	Model *F* value probability
MODEL_MEAN_SQUARE	Model mean square error
MODEL_SUM_SQUARES	Model sum of square errors
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)
NUM_ROWS	Number of rows
R_SQ	R-Square
RANK_DEFICIENCY	The number of predictors excluded from the model due to multi-collinearity
ROOT_MEAN_SQ	Root mean square error
SBIC	Schwarz's Bayesian information criterion

Global Details for GLM: Logistic Regression

The following table returns Global Name-Value Pairs (DM$VG) for a logistic regression model.

Table 36-33 Global Details for Logistic Regression

Name	Description
AIC_INTERCEPT	Akaike's criterion for the fit of the baseline, intercept-only, model
AIC_MODEL	Akaike's criterion for the fit of the intercept and the covariates (predictors) mode
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
DEPENDENT_MEAN	Dependent mean
ITERATIONS	Tracks the number of SGD iterations (number of IRLS iterations). Applicable only when the solver is SGD.
LR_DF	Likelihood ratio degrees of freedom
LR_CHI_SQ	Likelihood ratio chi-square value
LR_CHI_SQ_P_VALUE	Likelihood ratio chi-square probability value
NEG2_LL_INTERCEPT	-2 log likelihood of the baseline, intercept-only, model
NEG2_LL_MODEL	-2 log likelihood of the model

Table 36-33 (Cont.) Global Details for Logistic Regression

Name	Description
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)
NUM_ROWS	Number of rows
PCT_CORRECT	Percent of correct predictions
PCT_INCORRECT	Percent of incorrectly predicted rows
PCT_TIED	Percent of cases where the estimated probabilities are equal for both target classes
PSEUDO_R_SQ_CS	Pseudo R-square Cox and Snell
PSEUDO_R_SQ_N	Pseudo R-square Nagelkerke
RANK_DEFICIENCY	The number of predictors excluded from the model due to multi-collinearity
SC_INTERCEPT	Schwarz's Criterion for the fit of the baseline, intercept-only, model
SC_MODEL	Schwarz's Criterion for the fit of the intercept and the covariates (predictors) model

Note:

- When ridge regression is enabled, fewer global details are returned. For information about ridge, see *Oracle Machine Learning for SQL Concepts*.
- When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- Model Detail Views for Global Information
  Model detail views for global information contain information about global statistics, alerts, and computed settings.

36.8.8 Model Detail View for Multivariate State Estimation Technique - Sequential Probability Ratio Test

The model detail view specific to Multivariate State Estimation Technique - Sequential Probability Ratio Test contains information about Global Name-Value Paris.

The following are the available model views for MSET-SPRT:

Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs

Views	Description
`DM$VN`*model_name*	Normalization and Missing Value Handling
`DM$VS`*model_name*	Computed Settings
`DM$VT`*model_name*	Classification Targets
`DM$VW`*model_name*	Model Build Alerts

The following table lists the Global Name-Value Pairs (`DM$VG`*model_name*) for an MSET-SPRT. This statistic is included when due to memory constraints MSET-SPRT cannot use the `MSET_MEMORY_VECTORS` value set by the user.

Table 36-34 MSET-SPRT Information in the Model Global View

Name	Description
`NUM_MVEC`	The number of memory vectors used by the model.

36.8.9 Model Detail Views for Naive Bayes

The model detail views specific to Naive Bayes are the prior view and result view.

These the model views available for Naive Bayes:

Model Views	Description
`DM$VB`*model_name*	Automatic Data Preparation Binning
`DM$VC`*model_name*	Scoring Cost Matrix
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VP`*model_name*	Naive Bayes Target Priors
`DM$VS`*model_name*	Computed Settings
`DM$VT`*model_name*	Classification Targets
`DM$VV`*model_name*	Naive Bayes Conditional Probabilities
`DM$VW`*model_name*	Model Build Alerts

The Naive Bayes Target Priors view (`DM$VP`*model_name*) describes the priors of the targets for a Naive Bayes model. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_NAME                               VARCHAR2(128)
TARGET_VALUE                              NUMBER/VARCHAR2
PRIOR_PROBABILITY                         BINARY_DOUBLE
COUNT                                     NUMBER
```

Table 36-35 Naive Bayes Target Priors View for Naive Bayes

Column Name	Description
`PARTITION_NAME`	The name of a feature in the model

Table 36-35 (Cont.) Naive Bayes Target Priors View for Naive Bayes

Column Name	Description
TARGET_NAME	Name of the target column
TARGET_VALUE	Target value, numerical or categorical
PRIOR_PROBABILITY	Prior probability for a given TARGET_VALUE
COUNT	Number of rows for a given TARGET_VALUE

The Naive Bayes Conditional Probabilities view (DM$VV*model_view*) describes the conditional probabilities of the Naive Bayes model. The view has the following columns:

```
Name                                     Type
---------------------------------------- ----------------------------
PARTITION_NAME                           VARCHAR2(128)
TARGET_NAME                              VARCHAR2(128)
TARGET_VALUE                             NUMBER/VARCHAR2
ATTRIBUTE_NAME                           VARCHAR2(128)
ATTRIBUTE_SUBNAME                        VARCHAR2(4000)
ATTRIBUTE_VALUE                          VARCHAR2(4000)
CONDITIONAL_PROBABILITY                  BINARY_DOUBLE
COUNT                                    NUMBER
```

Table 36-36 Naive Bayes Conditional Probabilities View for Naive Bayes

Column Name	Description
PARTITION_NAME	The name of a feature in the model
TARGET_NAME	Name of the target column
TARGET_VALUE	Target value, numerical or categorical
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Machine learning attribute value for the column ATTRIBUTE_NAME or the nested column ATTRIBUTE_SUBNAME (if any).
CONDITIONAL_PROBABILITY	Conditional probability of a machine learning attribute for a given target
COUNT	Number of rows for a given machine learning attribute and a given target

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to a Naive Bayes model.

Table 36-37 Global Name-Value Pairs View for Naive Bayes

Name	Description
NUM_ROWS	The total number of rows used in the build

36.8.10 Model Detail Views for Neural Network

Model detail views specific to Neural Network contain information about the weights of the neurons: input layer and hidden layers.

These are the model views available for Neural Network:

Model Views	Description
DM$VA*model_name*	Neural Network Weights
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The Neural Network Weights view (DM$VA*model_name*) has the following columns:

```
Name
Type
----------------------          ----------------------
PARTITION_NAME                  VARCHAR2(128)
LAYER                           NUMBER
IDX_FROM                        NUMBER
ATTRIBUTE_NAME                  VARCHAR2(128)
ATTRIBUTE_SUBNAME               VARCHAR2(4000)
ATTRIBUTE_VALUE                 VARCHAR2(4000)
IDX_TO                          NUMBER
TARGET_VALUE                    NUMBER/VARCHAR2
WEIGHT                          BINARY_DOUBLE
```

Table 36-38 Neural Network Weights View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
LAYER	Layer ID, 0 as an input layer
IDX_FROM	Node index that the weight connects from (attribute id for input layer)
ATTRIBUTE_NAME	Attribute name (only for the input layer)
ATTRIBUTE_SUBNAME	Attribute subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
IDX_TO	Node index that the weights connects to
TARGET_VALUE	Target value. The value is null for regression.
WEIGHT	Value of the weight

The view Global Name-Value Pairs (`DM$VG`*model_name*) is a pre-existing view. The following name-value pairs are specific to a Neural Network view.

Table 36-39 Global Name-Value Pairs Viewfor Neural Network

Name	Description
`CONVERGED`	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • `YES` • `NO`
`ITERATIONS`	Number of iterations
`LOSS_VALUE`	Loss function value (if it is with `NNET_REGULARIZER_HELDASIDE` regularization, it is the loss function value on test data)
`NUM_ROWS`	Number of rows in the model (or partitioned model)

36.8.11 Model Detail Views for Random Forest

Model detail views specific to Random Forest contain variable importance measures and statistics.

The following model detail views are available for Random Forest:

Model View	Description
`DM$VA`*model_name*	Variable Importance
`DM$VC`*model_name*	Scoring Cost Matrix
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VS`*model_name*	Computed Settings
`DM$VT`*model_name*	Classification Targets
`DM$VW`*model_name*	Model Build Alerts

Model detail views and statistics specific to Random Forest are:

- Variable Importance statistics `DM$VA`*model_name*
- Random Forest statistics in the Global Name-Value Pairs `DM$VG`*model_name* view

One of the important outputs from a Random Forest model build is a ranking of attributes based on their relative importance. This is measured using Mean Decrease Gini. The `DM$VA`*model_name* view has the following columns:

```
Name                              Type
------------------------          ---------------
PARTITION_NAME                    VARCHAR2(128)
ATTRIBUTE_NAME                    VARCHAR2(128)
ATTRIBUTE_SUBNAME                 VARCHAR2(128)
ATTRIBUTE_IMPORTANCE              BINARY_DOUBLE
```

Table 36-40 Variable Importance Model View

Column Name	Description
PARTITION_NAME	Partition name. The value is null for models which are not partitioned.
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_IMPORTANCE	Measure of importance for an attribute in the forest (mean Decrease Gini value)

The Global Name-Value Pairs (DM$VG*model_name*) view is a pre-existing view. The following name-value pairs are added to the view.

Table 36-41 Random Forest Statistics Information In Model Global View

Name	Description
AVG_DEPTH	Average depth of the trees in the forest
AVG_NODECOUNT	Average number of nodes per tree
MAX_DEPTH	Maximum depth of the trees in the forest
MAX_NODECOUNT	Maximum number of nodes per tree
MIN_DEPTH	Minimum depth of the trees in the forest
MIN_NODECOUNT	Minimum number of nodes per tree
NUM_ROWS	The total number of rows used in the build

36.8.12 Model Detail View for Support Vector Machine

Model detail views specific to Support Vector Machine (SVM) contain linear coefficients and support vector statistics.

These model views are available for SVM:

Model Views	Description
DM$VCS*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The linear coefficient view DM$VL*model_name* describes the coefficients of a linear SVM algorithm. The *target_value* field in the view is present only for classification and has the type of the target. Regression models do not have a *target_value* field.

The *reversed_coefficient* field shows the value of the coefficient after reversing the automatic data preparation transformations. If data preparation is disabled, then *coefficient* and *reversed_coefficient* have the same value. The view has the following columns:

```
Name                                      Type
----------------------------------------- -------------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_VALUE                              NUMBER/VARCHAR2
ATTRIBUTE_NAME                            VARCHAR2(128)
ATTRIBUTE_SUBNAME                         VARCHAR2(4000)
ATTRIBUTE_VALUE                           VARCHAR2(4000)
COEFFICIENT                               BINARY_DOUBLE
REVERSED_COEFFICIENT                      BINARY_DOUBLE
```

Table 36-42 Linear Coefficient View for Support Vector Machine

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
TARGET_VALUE	Target value, numerical or categorical
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Value of a categorical attribute
COEFFICIENT	Projection coefficient value
REVERSED_COEFFICIENT	Coefficient transformed on the original scale

The following table describes the SVM statistics global view.

Table 36-43 Support Vector Statistics Information In Model Global View

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance: • YES • NO
ITERATIONS	Number of iterations performed during build
NUM_ROWS	Number of rows used for the build
REMOVED_ROWS_ZERO_NORM	Number of rows removed due to 0 norm. This applies to one-class linear models only.

36.8.13 Model Detail Views for XGBoost

The model detail views specific to XGBoost contain information about Feature Importance view and Global Name-Value Pairs view.

The following are the available model views for XGBoost Classification:

Model Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	XGBoost Attribute Importance
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The following are the available model views for XGBoost Regression:

Views	Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	XGBoost Attribute Importance
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The `DM$VI`*`model_name`* view reports the feature importance values for each attribute of each partition of the model.

The view has the following columns for tree models (`gbtree` and `dart` boosters).

```
Name                 Type
-----------------    --------------
PNAME                VARCHAR2(128)
ATTRIBUTE_NAME       VARCHAR2(128)
ATTRIBUTE_SUBNAME    VARCHAR2(4000)
ATTRIBUTE_VALUE      VARCHAR2(4000)
GAIN                 BINARY_DOUBLE
COVER                BINARY_DOUBLE
FREQUENCY            BINARY_DOUBLE
```

Table 36-44 Feature Importance View for a Tree Model

Column Name	Description
PNAME	The name of a partition in a partitioned model.
ATTRIBUTE_NAME	The column name.
ATTRIBUTE_SUBNAME	The nested column subname; the value is null for non-nested columns.
ATTRIBUTE_VALUE	The value of a categorical attribute.
GAIN	The fractional contribution of each feature to the model based on the total gain of a feature's splits; a higher percentage means a more important predictive feature.
COVER	The number of observation either seen by a split or collected by a leaf during training.
FREQUENCY	A percentage representing the relative number of times a feature has been used in trees.

For a linear model (`gblinear`) booster, the feature importance is the absolute magnitude of linear coefficients.

The view has the following columns for linear models.

```
Name            Type
-----------------  --------------
PNAME              VARCHAR2(128)
ATTRIBUTE_NAME     VARCHAR2(128)
ATTRIBUTE_SUBNAME  VARCHAR2(4000)
ATTRIBUTE_VALUE    VARCHAR2(4000)
WEIGHT             BINARY_DOUBLE
CLASS              BINARY_DOUBLE
```

Table 36-45 Feature Importance View for a Linear Model

Column Name	Description
`PNAME`	The name of a partition in a partitioned model.
`ATTRIBUTE_NAME`	The column name.
`ATTRIBUTE_SUBNAME`	The nested column subname; the value is null for non-nested columns.
`ATTRIBUTE_VALUE`	The value of a categorical attribute.
`WEIGHT`	The linear coefficient of the feature.
`CLASS`	The class label for a multiclass model.

The `DM$VG`*`model_name`* view reports global statistics for an XGBoost model. The statistics include an evaluation of the training data set using the evaluation metric you specified with the learning task `eval_metric` setting, or the default `eval_metric` if you didn't specify one. The view displays only the result of the last training iteration. When you specify more than one `eval_metric`, the view contains multiple rows, one for each `eval_metric`.

36.8.14 Model Detail Views for Clustering Algorithms

Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

All clustering algorithms share the following views:

Model Views	Description
`DM$VD`*model_name*:	Clustering Description
`DM$VA`*model_name*	Clustering Attribute Statistics
`DM$VH`*model_name*	Clustering Histograms
`DM$VR`*model_name*	Clustering Rules

The Cluster Description view `DM$VD`*model_name* describes cluster level information about a clustering model. The view has the following columns:

```
Name                                  Type
----------------------------------  ----------------------------
PARTITION_NAME                        VARCHAR2(128)
```

```
CLUSTER_ID                          NUMBER
CLUSTER_NAME                        NUMBER/VARCHAR2
RECORD_COUNT                        NUMBER
PARENT                              NUMBER
TREE_LEVEL                          NUMBER
LEFT_CHILD_ID                       NUMBER
RIGHT_CHILD_ID                      NUMBER
```

Table 36-46 Clustering Description View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
RECORD_COUNT	Specifies the number of records
PARENT	The ID of the parent
TREE_LEVEL	Specifies the number of splits from the root
LEFT_CHILD_ID	The ID of the child cluster on the left side of the split
RIGHT_CHILD_ID	The ID of the child cluster on the right side of the split

The attribute view `DM$VA`*model_name* describes attribute level information about a clustering model. The values of the mean, variance, and mode for a particular cluster can be obtained from this view. The view has the following columns:

```
Name                                Type
----------------------------------  ----------------------------
PARTITION_NAME                      VARCHAR2(128)
CLUSTER_ID                          NUMBER
CLUSTER_NAME                        NUMBER/VARCHAR2
ATTRIBUTE_NAME                      VARCHAR2(128)
ATTRIBUTE_SUBNAME                   VARCHAR2(4000)
MEAN                                BINARY_DOUBLE
VARIANCE                            BINARY_DOUBLE
MODE_VALUE                          VARCHAR2(4000)
```

Table 36-47 Clustering Attribute Statistics

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
MEAN	The field returns the average value of a numeric attribute
VARIANCE	The variance of a numeric attribute

Table 36-47 (Cont.) Clustering Attribute Statistics

Column Name	Description
MODE_VALUE	The mode is the most frequent value of a categorical attribute

The histogram view DM$VH*model_name* describes histogram level information about a clustering model. The bin information as well as bin counts can be obtained from this view. The view has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
CLUSTER_ID                            NUMBER
CLUSTER_NAME                          NUMBER/VARCHAR2
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
BIN_ID                                NUMBER
LOWER_BIN_BOUNDARY                    BINARY_DOUBLE
UPPER_BIN_BOUNDARY                    BINARY_DOUBLE
ATTRIBUTE_VALUE                       VARCHAR2(4000)
COUNT                                 NUMBER
```

Table 36-48 Clustering Histograms View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
BIN_ID	Bin ID
LOWER_BIN_BOUNDARY	Numeric lower bin boundary
UPPER_BIN_BOUNDARY	Numeric upper bin boundary
ATTRIBUTE_VALUE	Categorical attribute value
COUNT	Histogram count

The rule view DM$VR*model_name* describes the rule level information about a clustering model. The information is provided at attribute predicate level. The view has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
CLUSTER_ID                            NUMBER
CLUSTER_NAME                          NUMBER/VARCHAR2
ATTRIBUTE_NAME                        VARCHAR2(128)
```

```
ATTRIBUTE_SUBNAME                  VARCHAR2(4000)
OPERATOR                           VARCHAR2(2)
NUMERIC_VALUE                      NUMBER
ATTRIBUTE_VALUE                    VARCHAR2(4000)
SUPPORT                            NUMBER
CONFIDENCE                         BINARY_DOUBLE
RULE_SUPPORT                       NUMBER
RULE_CONFIDENCE                    BINARY_DOUBLE
```

Table 36-49   Clustering Rules View

Column Name	Description
`PARTITION_NAME`	A partition in a partitioned model
`CLUSTER_ID`	The ID of a cluster in the model
`CLUSTER_NAME`	Specifies the label of the cluster
`ATTRIBUTE_NAME`	Specifies the attribute name
`ATTRIBUTE_SUBNAME`	Specifies the attribute subname
`OPERATOR`	Attribute predicate operator - a conditional operator taking the following values: *IN*, = , <>, < , >, <=, and >=
`NUMERIC_VALUE`	Numeric lower bin boundary
`ATTRIBUTE_VALUE`	Categorical attribute value
`SUPPORT`	Attribute predicate support
`CONFIDENCE`	Attribute predicate confidence
`RULE_SUPPORT`	Rule level support
`RULE_CONFIDENCE`	Rule level confidence

36.8.15 Model Detail Views for Expectation Maximization

Model detail views specific to Expectation Maximization (EM) contain additional information about an EM model. Additional views are available for EM Clustering, but are absent for EM Anomaly.

These are the model views available for Expectation Maximization:

Model Views	Description
`DM$VA`*model_name*	Clustering Attribute Statistics
`DM$VB`*model_name*	Attribute Pair Kullback-Leibler Divergence
`DM$VD`*model_name*	Clustering Description
`DM$VF`*model_name*	Expectation Maximization Bernoulli parameters
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VH`*model_name*	Clustering Histograms
`DM$VI`*model_name*	Unsupervised Attribute Importance
`DM$VM`*model_name*	Expectation Maximization Gaussian parameters
`DM$VN`*model_name*	Normalization and Missing Value Handling

Model Views	Description
DM$VO*model_name*	Expectation Maximization Components
DM$VP*model_name*	Expectation Maximization Projections
DM$VR*model_name*	Clustering Rules
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

For EM Clustering model, the following views contain information that is not in the clustering views. For the clustering views, refer to "Model Detail Views for Clustering Algorithms".

The Expectation Maximization Components view (DM$VO*model_name*) describes the EM Cluster components. The component view contains information about their prior probabilities and what cluster they map to. The view has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
COMPONENT_ID                          NUMBER
CLUSTER_ID                            NUMBER
PRIOR_PROBABILITY                     BINARY_DOUBLE
```

Table 36-50 Expectation Maximization Components View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
COMPONENT_ID	Unique identifier of a component
CLUSTER_ID	The ID of a cluster in the model
PRIOR_PROBABILITY	Component prior probability

The Expectation Maximization Gaussian view (DM$VM*model_name*) provides information about the mean and variance parameters for the attributes by Gaussian distribution models. The view has the following columns:

```
Name                                  Type
------------------------------------- ----------------------------
PARTITION_NAME                        VARCHAR2(128)
COMPONENT_ID                          NUMBER
ATTRIBUTE_NAME                        VARCHAR2(4000)
MEAN                                  BINARY_DOUBLE
VARIANCE                              BINARY_DOUBLE
```

The Expectation Maximization Bernoulli parameters view (DM$VF*model_name*) provides information about the parameters of the multi-valued Bernoulli distributions used by the EM model. The view has the following columns:

```
Name                                   Type
 ------------------------------------- ----------------------------
 PARTITION_NAME                        VARCHAR2(128)
```

```
COMPONENT_ID                              NUMBER
ATTRIBUTE_NAME                            VARCHAR2(4000)
ATTRIBUTE_VALUE                           VARCHAR2(4000)
FREQUENCY                                 BINARY_DOUBLE
```

Table 36-51 Expectation Maximization Bernoulli parameters View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
COMPONENT_ID	Unique identifier of a component
ATTRIBUTE_NAME	Column name
ATTRIBUTE_VALUE	Categorical attribute value
FREQUENCY	The frequency of the multivalued Bernoulli distribution for the attribute/value combination specified by ATTRIBUTE_NAME and ATTRIBUTE_VALUE.

For 2-Dimensional columns, EM provides an attribute ranking similar to that of attribute importance. This ranking is based on a rank-weighted average over Kullback–Leibler divergence computed for pairs of columns. This unsupervised attribute importance is shown in the Unsupervised Attribute Importance view (DM$VI*model_name*) and has the following columns:

```
Name                                       Type
----------------------------------------- ----------------------------
PARTITION_NAME                             VARCHAR2(128)
ATTRIBUTE_NAME                             VARCHAR2(128)
ATTRIBUTE_IMPORTANCE_VALUE                 BINARY_DOUBLE
ATTRIBUTE_RANK                             NUMBER
```

Table 36-52 Unsupervised Attribute Importance View for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_IMPORTANCE_VALUE	Importance value
ATTRIBUTE_RANK	An attribute rank based on the importance value

The pairwise Kullback–Leibler divergence is reported in the Attribute Pair Kullback-Leibler Divergence view (DM$VB*model_name*). This metric evaluates how much the observed joint distribution of two attributes diverges from the expected distribution under the assumption of independence. That is, the higher the value, the more dependent the two attributes are. The dependency value is scaled based on the size of the grid used for each pairwise computation. That ensures that all values fall within the [0; 1] range and are comparable. The view has the following columns:

```
Name                                       Type
----------------------------------------- ----------------------------
PARTITION_NAME                             VARCHAR2(128)
```

```
ATTRIBUTE_NAME_1                          VARCHAR2(128)
ATTRIBUTE_NAME_2                          VARCHAR2(128)
DEPENDENCY                                BINARY_DOUBLE
```

Table 36-53 Attribute Pair Kullback-Leibler Divergence View for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME_1	Name of the first attribute
ATTRIBUTE_NAME_2	Name of the second attribute
DEPENDENCY	Scaled pairwise Kullback-Leibler divergence

The projection table `DM$VP`*model_name* shows the coefficients used by random projections to map nested columns to a lower dimensional space. The view has rows only when nested or text data is present in the build data. The view has the following columns:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
FEATURE_NAME                          VARCHAR2(4000)
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
ATTRIBUTE_VALUE                       VARCHAR2(4000)
COEFFICIENT                           NUMBER
```

Table 36-54 Projection table for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_NAME	Name of feature
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	Projection coefficient. The representation is sparse; only the non-zero coefficients are returned.

For EM Anomaly, currently there are no additional views other than the classification views. For the classification view, refer to “Model Detail Views for Classification Algorithms”.

Global Details for Expectation Maximization

The following table describes global details for EM.

Table 36-55 Global Details for Expectation Maximization

Name	Description
`CONVERGED`	Indicates whether the model build process has converged to specified tolerance. The possible values are: • `YES` • `NO`
`LOGLIKELIHOOD`	Loglikelihood on the build data
`NUM_COMPONENTS`	Number of components produced by the model
`NUM_CLUSTERS`	Number of clusters produced by the model (only available for EM Clustering)
`NUM_ROWS`	Number of rows used in the build
`RANDOM_SEED`	The random seed value used for the model build
`REMOVED_COMPONENTS`	The number of empty components excluded from the model

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

36.8.16 Model Detail Views for *k*-Means

Model detail views specific to *k*-Means (KM) contain clustering description view (`DM$VG`), and scoring information.

The following model views are available for *k*-Means algorithm.

Model Views	Description
`DM$VA`*model_name*	Clustering Attribute Statistics
`DM$VC`*model_name*	k-Means Scoring Centroids
`DM$VD`*model_name*	Clustering Description
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VH`*model_name*	Clustering Histograms
`DM$VN`*model_name*	Normalization and Missing Value Handling
`DM$VR`*model_name*	Clustering Rules
`DM$VS`*model_name*	Computed Settings
`DM$VW`*model_name*	Model Build Alerts

"Model Detail Views for Clustering Algorithms" discusses common model views across clustering algorithms. Global Name-Value Pairs view (`DM$VG`), which contains information about Computed Settings view (`DM$VS`) and Model Build Alerts view (`DM$VW`), and Normalization and Missing Value Handling view (`DM$VN`) are addressed individually.

The following views contain information that is specific to *k*-Means model.

The *k*-Means Clustering Description view `DM$VD`*model_name* has an additional column:

```
Name                                  Type
----------------------------------    ----------------------------
DISPERSION                            BINARY_DOUBLE
```

Table 36-56 Clustering Description for k-Means

Column Name	Description
DISPERSION	A measure used to quantify whether a set of observed occurrences are dispersed compared to a standard statistical model.

The *k*-Means Scoring Centroids view `DM$VC`*model_name* describes the centroid of each leaf clusters:

```
Name                                   Type
 ----------------------------------    ----------------------------
 PARTITION_NAME                        VARCHAR2(128)
 CLUSTER_ID                            NUMBER
 CLUSTER_NAME                          NUMBER/VARCHAR2
 ATTRIBUTE_NAME                        VARCHAR2(128)
 ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
 ATTRIBUTE_VALUE                       VARCHAR2(4000)
 VALUE                                 BINARY_DOUBLE
```

Table 36-57 k-Means Scoring Centroids View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
VALUE	Specifies the centroid value

The following table describes Global Name-Value Pairs view (`DM$VG`) for *k*-Means.

Table 36-58 *k*–Means Global Name-Value Pairs View

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO

Table 36-58 (Cont.) *k*–Means Global Name-Value Pairs View

Name	Description
NUM_ROWS	Number of rows used in the build
REMOVED_ROWS_ZERO_NORM	Number of rows removed due to 0 norm. This applies only to models using cosine distance.

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).
- Model Detail Views for Global Information
  Model detail views for global information contain information about global statistics, alerts, and computed settings.

36.8.17 Model Detail Views for O-Cluster

Model detail views specific to O-Cluster (OC) contain information about description view, histograms view, and global view.

These are the available model views for O-Cluster:

Model Views	Description
DM$VA*model_name*	Clustering Attribute Statistics
DM$VB*model_name*	Automatic Data Preparation Binning
DM$VD*model_name*	Clustering Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VH*model_name*	Clustering Histograms
DM$VR*model_name*	Clustering Rules
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The following views contain information that is specific to an O-Cluster model. For the clustering views, refer to "Model Detail Views for Clustering Algorithms". The OC algorithm uses the same descriptive statistics views as Expectation Maximization (EM) and *k*-Means (KM). The following are the statistics views:

The Cluster Description view (DM$VD*model_name*) describes the O-Cluster components. The Cluster Description view has additional fields that specify the split predicate. The view has the following columns:

```
Name                                  Type
------------------------------------ ----------------------------
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
OPERATOR                              VARCHAR2(2)
VALUE                                 SYS.XMLTYPE
```

Table 36-59 Cluster Description View for O-Cluster

Column Name	Description
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
OPERATOR	Split operator
VALUE	List of split values

The structure of the SYS.XMLTYPE is as follows:

```
<Element>splitval1</Element>
```

The OC algorithm uses a Clustering Histograms view (DM$VH*model_name*) with different columns than EM and KM. The view has the following columns:

```
Name                                  Type
------------------------------------- ------------------------------
PARTITON_NAME                         VARCHAR2(128)
CLUSTER_ID                            NUMBER
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
BIN_ID                                NUMBER
LABEL                                 VARCHAR2(4000)
COUNT                                 NUMBER
```

Table 36-60 Clustering Histograms View for O-Cluster

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CLUSTER_ID	Unique identifier of a component
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
BIN_ID	Unique identifier
LABEL	Bin label
COUNT	Bin histogram count

The following table describes the Global Name-Value Pairs (DM$VG*model_name*) view specific to O-Cluster.

Table 36-61 O-Cluster Statistics Information In Model Global View

Name	Description
NUM_ROWS	The total number of rows used in the build

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

36.8.18 Model Detail Views for CUR Matrix Decomposition

Model detail views for CUR Matrix Decomposition contain information about the scores and ranks of attributes and rows.

CUR Matrix Decomposition models have the following views:

Attribute importance and rank: `DM$VC`*model_name*

Row importance and rank: `DM$VR`*model_name*

Global statistics: `DM$VG`

The attribute importance and rank view `DM$VC`*model_name* has the following columns:

```
Name                          Type
-----------------             -----------------
PARTITION_NAME                VARCHAR2(128)
ATTRIBUTE_NAME                VARCHAR2(128)
ATTRIBUTE_SUBNAME             VARCHAR2(4000)
ATTRIBUTE_VALUE               VARCHAR2(4000)
ATTRIBUTE_IMPORTANCE          NUMBER
ATTRIBUTE_RANK                NUMBER
```

Table 36-62   Attribute Importance and Rank View

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`ATTRIBUTE_NAME`	Attribute name
`ATTRIBUTE_SUBNAME`	Attribute subname. The value is null for non-nested columns.
`ATTRIBUTE_VALUE`	Value of the attribute
`ATTRIBUTE_IMPORTANCE`	Attribute leverage score
`ATTRIBUTE_RANK`	Attribute rank based on leverage score

The view `DM$VR`*model_name* exposes the leverage scores and ranks of all selected rows through a view. This view is created when users decide to perform row importance and the `CASE_ID` column is present. The view has the following columns:

```
Name                       Type
---------------------         ------------------------
PARTITION_NAME                VARCHAR2(128)
CASE_ID                       Original cid data types,
                              including NUMBER, VARCHAR2,
                              DATE, TIMESTAMP,
                              TIMESTAMP WITH TIME ZONE,
```

```
                              TIMESTAMP WITH LOCAL TIME ZONE
ROW_IMPORTANCE                NUMBER
ROW_RANK                      NUMBER
```

Table 36-63 Row Importance and Rank View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Case ID. The supported case ID types are the same as that supported for GLM, SVD, and ESA algorithms.
ROW_IMPORTANCE	Row leverage score
ROW_RANK	Row rank based on leverage score

The following table describes global statistics for CUR Matrix Decomposition.

Table 36-64 CUR Matrix Decomposition Statistics Information In Model Global View.

Name	Description
NUM_COMPONENTS	Number of SVD components (SVD rank)
NUM_ROWS	Number of rows used in the model build

36.8.19 Model Detail Views for Explicit Semantic Analysis

Model detail views specific to Explicit Semantic Analysis (ESA) contain information about attribute statistics and features.

These are the available model views:

Model Views	Description
DM$VA*model_name*	Explicit Semantic Analysis Matrix
DM$VF*model_name*	Explicit Semantic Analysis Features
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts
DM$VX*model_name*	Text Features

- Explicit Semantic Analysis Matrix (DM$VA*model_name*): This view has different columns for feature extraction and classification. For feature extraction, this view contains model attribute coefficients per feature. For classification, this view contains model attribute coefficients per target class.
- Explicit Semantic Analysis Features (DM$VF*model_name*): This view is applicable only for feature extraction.

The Explicit Semantic Analysis Matrix view (DM$VA*model_name*) has the following columns for feature extraction:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
FEATURE_ID                            NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                      TIMESTAMP WITH TIME ZONE,
                                      TIMESTAMP WITH LOCAL TIME ZONE
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
ATTRIBUTE_VALUE                       VARCHAR2(4000)
COEFFICIENT                           BINARY_DOUBLE
```

Table 36-65 Explicit Semantic Analysis Matrix for Feature Extraction

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	Unique identifier of a feature as it appears in the training data
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	A measure of the weight of the attribute with respect to the feature

The (DM$VA*model_name*) view comprises of attribute coefficients for all target classes.

The view Explicit Semantic Analysis Matrix (DM$VA*model_name*) has the following columns for classification:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
TARGET_VALUE                          NUMBER/VARCHAR2
ATTRIBUTE_NAME                        VARCHAR2(128)
ATTRIBUTE_SUBNAME                     VARCHAR2(4000)
ATTRIBUTE_VALUE                       VARCHAR2(4000)
COEFFICIENT                           BINARY_DOUBLE
```

Table 36-66 Explicit Semantic Analysis Matrix for Classification

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
TARGET_VALUE	Value of the target
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.

Table 36-66 (Cont.) Explicit Semantic Analysis Matrix for Classification

Column Name	Description
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	A measure of the weight of the attribute with respect to the feature

The Explicit Semantic Analysis Features view (DM$VF*model_name*) has a unique row for every feature in one view. This feature is helpful if the model was pre-built and the source training data are not available. The view has the following columns:

```
Name                                   Type
----------------------------------     ----------------------------
PARTITION_NAME                         VARCHAR2(128)
FEATURE_ID                             NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                       TIMESTAMP WITH TIME ZONE,
                                       TIMESTAMP WITH LOCAL TIME ZONE
```

Table 36-67 Explicit Semantic Analysis Features for Explicit Semantic Analysis

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	Unique identifier of a feature as it appears in the training data

The following table describes the Global Name-Value Pairs view (DM$VG*model_name)* specific to ESA.

Table 36-68 Explicit Semantic Analysis Statistics Information In Model Global View

Name	Description
NUM_ROWS	The total number of input rows
REMOVED_ROWS_BY_FILTERS	Number of rows removed by filters

36.8.20 Model Detail Views for Exponential Smoothing

Model detail views specific to Exponential Smoothing (ESM) include information about the model output, global information about the model, and views that support time series regression.

These are the available model views for ESM:

Model Details	Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VP*model_name*	Exponential Smoothing Forecast
DM$VS*model_name*	Computed Settings

Model Details	Description
DM$VW*model_name*	Model Build Alerts
DM$VR*model_name*	Time Series Regression Build
DM$VT*model_name*	Time Series Regression Score

Exponential Smoothing Forecast view (DM$VP*model_name*) displays the outcome of an ESM model. The output contains a set of records, ordered by partition and CASE_ID, that include the columns given in the *Exponential Smoothing Model Output* table. CASE_ID identifies the value's position in the time series. The user-specified CASE_ID can be a type that represents a numerical or datetime value. For each unique value of PARTITION, a distinct exponential smoothing model is built. The VALUE column for each PARTITION represents the observed or accumulated value of the target at that point in the sequence. The PREDICTION column is the forecast one step ahead at that point in the sequence. Backcasts are predictions that fall inside the range of the input data. The sequence also includes a user-specified number of values beyond the range of the input data. The VALUE column is *NULL* for any sequence value outside the range of input, and PREDICTION column is the model forecast for that sequence value. Lower and upper boundaries of the forecasts are denoted by the LOWER and UPPER columns. For backcasts, LOWER and UPPER are *NULL*. The bounds are based on a confidence interval that the user sets for the prediction.

Table 36-69 Exponential Smoothing Forecast View

Name	Description
PARTITION	Partition name in a partitioned model
CASE_ID	Sequence identifier (datetime or number type)
VALUE	Observed or accumulated value
PREDICTION	Backcast or Forecast value
UPPER	Upper bound of the forecast
LOWER	Lower bound of the forecast

Global Name-Value Pairs view (DM$VG*model_name*) includes the model's global information as well as the estimated smoothing constants, estimated initial state, and global diagnostic measures.

Depending on the type of model, the global diagnostics include some or all of the following for Exponential Smoothing.

Table 36-70 Global Name-Value Pairs View for ESM

Name	Description
-2 LOG-LIKELIHOOD	Negative log-likelihood of model
ALPHA	Smoothing constant
AIC	Akaike information criterion
AICC	Corrected Akaike information criterion

Table 36-70 (Cont.) Global Name-Value Pairs View for ESM

Name	Description
`AMSE`	Average mean square error over user-specified time window
`BETA`	Trend smoothing constant
`BIC`	Bayesian information criterion
`GAMMA`	Seasonal smoothing constant
`INITIAL LEVEL`	Model estimate of value one time interval prior to start of observed series
`INITIAL SEASON i`	Model estimate of seasonal effect for season *i* one time interval prior to start of observed series
`INITIAL TREND`	Model estimate of trend one time interval prior to start of observed series
`MAE`	Model mean absolute error
`MSE`	Model mean square error
`PHI`	Damping parameter
`STD`	Model standard error
`SIGMA`	Model standard deviation of residuals

Time series regression expands the features that can be included in a time series model and, possibly, increases forecast accuracy. Backcasts and forecasts of time series correlated to the "target" series of interest are included in the build and score views. The build and score views can be fed into a regression technique like Generalized Linear Model.

The Time Series Regression Build view (`DM$VR`*model_name*) depicts the schema for the build view. Each predictor series will have its own column. There can be a maximum of 20 predictor series in the build and score views. The names of the columns are obtained from the `EXSM_SERIES_LIST` setting.

Table 36-71 Time Series Regression Build View

Name	Description
`PARTITION`	Partition name in a partitioned model
`CASE_ID`	Sequence identifier (datetime or number type)
`target series name`	Observed or accumulated value of target series
`DM$`*`target series`*	Backcasted value of target series
`DM$`*`predictor series column name`*	Backcasted value of predictor series column. A maximum of *20* predictor series columns can be used.

The Time Series Regression Score view (`DM$VT`*model_name*) shows the schema for the score view. The schema is the same as in the build view, but the values in the *target series name* column are `NULL` because the future has not yet been observed.

Table 36-72 Time Series Regression Score View

Name	Description
`PARTITION`	Partition name in a partitioned model
`CASE_ID`	Sequence identifier (datetime or number type)
`target series name`	`NULL`s, because the future values of the target series have not been observed
`DM$`*`target series`*	Forecasted value of target series
`DM$`*`predictor series column name`*	Forecasted value of predictor series column name. A maximum of *20* predictor series columns can be used.

Related Topics

- About Exponential Smoothing
- About Generalized Linear Models

36.8.21 Model Detail Views for Non-Negative Matrix Factorization

Model detail views specific to Non-Negative Matrix Factorization (NMF) contain information about the encoding H matrix and H inverse matrix.

These are the available model views for NMF:

Model Views	Description
`DM$VE`*model_name*	Non-Negative Matrix Factorization H Matrix
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VI`*model_name*	Non-Negative Matrix Factorization Inverse H Matrix
`DM$VN`*model_name*	Normalization andMissing Value Handling
`DM$VS`*model_name*	Computed Settings
`DM$VW`*model_name*	Model Build Alerts

The views specific to NMF are:

- Non-Negative Matrix Factorization H Matrix view (`DM$VE`*model_name*)
- Non-Negative Matrix Factorization Inverse H Matrix view (`DM$VI`*model_name*)

The view `DM$VE`*model_name* describes the encoding (H) matrix of an NMF model. The `FEATURE_NAME` column type may be either `NUMBER` or `VARCHAR2`. The view has the following columns.

```
Name                 Type
-------------------  --------------------------
PARTITION_NAME       VARCHAR2(128)
FEATURE_ID           NUMBER
FEATURE_NAME         NUMBER/VARCHAR2
ATTRIBUTE_NAME       VARCHAR2(128)
ATTRIBUTE_SUBNAME    VARCHAR2(4000)
```

```
ATTRIBUTE_VALUE       VARCHAR2(4000)
COEFFICIENT           BINARY_DOUBLE
```

Table 36-73 Non-Negative Matrix Factorization H Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Specifies the value of attribute
COEFFICIENT	The attribute encoding that represents its contribution to the feature

The view `DM$VI`*model_view* describes the inverse H matrix of an NMF model. The `FEATURE_NAME` column type may be either `NUMBER` or `VARCHAR2`. The view has the following schema:

```
Name                 Type
-----------------    ------------------------
PARTITION_NAME       VARCHAR2(128)
FEATURE_ID           NUMBER
FEATURE_NAME         NUMBER/VARCHAR2
ATTRIBUTE_NAME       VARCHAR2(128)
ATTRIBUTE_SUBNAME    VARCHAR2(4000)
ATTRIBUTE_VALUE      VARCHAR2(4000)
COEFFICIENT          BINARY_DOUBLE
```

Table 36-74 Non-Negative Matrix Factorization Inverse H Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Specifies the value of attribute
COEFFICIENT	The attribute encoding that represents its contribution to the feature

The following table describes the Global Name-Value Pairs view (`DM$VG`*model_name*) specific to NMF.

Table 36-75 Global Name-Value Pairs View for NMF

Name	Description
`CONV_ERROR`	Convergence error
`CONVERGED`	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • `YES` • `NO`
`ITERATIONS`	Number of iterations performed during build
`NUM_ROWS`	Number of rows used in the build input data set
`SAMPLE_SIZE`	Number of rows used by the build

36.8.22 Model Detail Views for Singular Value Decomposition

Model detail views specific to Singular Value Decomposition (SVD) contain information about the S matrix, right-singular vectors, and left-singular vectors.

These are the available model views for SVD:

Model Views	Description
`DM$VE`*model_name*	Singular Value Decomposition S Matrix
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VN`*model_name*	Normalization and Missing Value Handling
`DM$VS`*model_name*	Computed Settings
`DM$VU`*model_name*	Singular Value Decomposition U Matrix
`DM$VV`*model_name*	Singular Value Decomposition V Matrix
`DM$VW`*model_name*	Model Build Alerts

The Singular Value Decomposition S Matrix view (`DM$VE`*model_name*) leverages the fact that each singular value in the SVD model has a corresponding principal component in the associated Principal Components Analysis (PCA) model to relate a common set of information for both classes of models. For an SVD model, it describes the content of the S matrix. When PCA scoring is selected as a build setting, the variance and percentage cumulative variance for the corresponding principal components are shown as well. The view has the following columns:

```
Name                                  Type
----------------------------------  ----------------------------
PARTITION_NAME                        VARCHAR2(128)
FEATURE_ID                            NUMBER
FEATURE_NAME                          NUMBER/VARCHAR2
VALUE                                 BINARY_DOUBLE
VARIANCE                              BINARY_DOUBLE
PCT_CUM_VARIANCE                      BINARY_DOUBLE
```

Table 36-76 Singular Value Decomposition S Matrix View

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`FEATURE_ID`	The ID of a feature in the model
`FEATURE_NAME`	The name of a feature in the model
`VALUE`	The matrix entry value
`VARIANCE`	The variance explained by a component. This column is only present for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` This column is non-null only if the build data is centered, either manually or because of the following setting:`dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.
`PCT_CUM_VARIANCE`	The percent cumulative variance explained by the components thus far. The components are ranked by the explained variance in descending order. This column is only present for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` This column is non-null only if the build data is centered, either manually or because of the following setting:`dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.

The Singular Value Decomposition V Matrix view (`DM$VV`*model_view*) describes the right-singular vectors of an SVD model. For a PCA model it describes the principal components (eigenvectors). The view has the following columns:

```
Name                                Type
----------------------------------  ----------------------------
PARTITION_NAME                      VARCHAR2(128)
FEATURE_ID                          NUMBER
FEATURE_NAME                        NUMBER/VARCHAR2
ATTRIBUTE_NAME                      VARCHAR2(128)
ATTRIBUTE_SUBNAME                   VARCHAR2(4000)
ATTRIBUTE_VALUE                     VARCHAR2(4000)
VALUE                               BINARY_DOUBLE
```

Table 36-77 Singular Value Decomposition V Matrix View

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`FEATURE_ID`	The ID of a feature in the model
`FEATURE_NAME`	The name of a feature in the model
`ATTRIBUTE_NAME`	Column name

Table 36-77 (Cont.) Singular Value Decomposition V Matrix View

Column Name	Description
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value. For numerical attributes, ATTRIBUTE_VALUE is null.
VALUE	The matrix entry value

The Singular Value Decomposition U Matrix view (DM$VU*model_name*) describes the left-singular vectors of an SVD model. For a PCA model, it describes the projection of the data in the principal components. This view does not exist unless the settings dbms_data_mining.svds_u_matrix_output is set to dbms_data_mining.svds_u_matrix_enable. The view has the following columns:

```
Name                                  Type
----------------------------------    ----------------------------
PARTITION_NAME                        VARCHAR2(128)
CASE_ID                               NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                      TIMESTAMP WITH TIME ZONE,
                                      TIMESTAMP WITH LOCAL TIME ZONE
FEATURE_ID                            NUMBER
FEATURE_NAME                          NUMBER/VARCHAR2
VALUE                                 BINARY_DOUBLE
```

Table 36-78 Singular Value Decomposition U Matrix View or Projection Data in Principal Components

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Unique identifier of the row in the build data described by the **U** matrix projection.
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
VALUE	The matrix entry value

Global Details for Singular Value Decomposition

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to a SVD model.

Table 36-79 Global Name-Value Pairs View for Singular Value Decomposition

Name	Description
NUM_COMPONENTS	Number of features (components) produced by the model
NUM_ROWS	The total number of rows used in the build

Table 36-79 (Cont.) Global Name-Value Pairs View for Singular Value Decomposition

Name	Description
SUGGESTED_CUTOFF	Suggested cutoff that indicates how many of the top computed features capture most of the variance in the model. Using only the features below this cutoff would be a reasonable strategy for dimensionality reduction.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

36.8.23 Model Detail Views for Minimum Description Length

Model detail views specific to Minimum Description Length (MDL) (for calculating attribute importance) contain information about attribute importance models.

These are the available model views for MDL:

Model Views	Description
DM$VA*model_name*	Attribute Importance
DM$VB*model_name*	Automatic Data Preparation Binning
DM$VG*model_name*	Global Name-Value Pairs
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The Attribute Importance view (DM$VA*model_name*) describes the attribute importance as well as the attribute importance rank. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
ATTRIBUTE_NAME                            VARCHAR2(128)
ATTRIBUTE_SUBNAME                         VARCHAR2(4000)
ATTRIBUTE_IMPORTANCE_VALUE                BINARY_DOUBLE
ATTRIBUTE_RANK                            NUMBER
```

Table 36-80 Attribute Importance View for Minimum Description Length

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_IMPORTANCE_VALUE	Importance value
ATTRIBUTE_RANK	Rank based on importance

The following table describes the Global Name-Value Pairs view (`DM$VG`*model_name*) specific to MDL.

Table 36-81 Global Name-Value Pairs View for MDL

Name	Description
`NUM_ROWS`	The total number of rows used in the build

36.8.24 Model Detail Views for Binning

The binning view `DM$VB` describes the bin boundaries used in automatic data preparation.

The view has the following columns:

```
Name                       Type
--------------------       --------------------
PARTITION_NAME             VARCHAR2(128)
ATTRIBUTE_NAME             VARCHAR2(128)
ATTRIBUTE_SUBNAME          VARCHAR2(4000)
BIN_ID                     NUMBER
LOWER_BIN_BOUNDARY         BINARY_DOUBLE
UPPER_BIN_BOUNDARY         BINARY_DOUBLE
ATTRIBUTE_VALUE            VARCHAR2(4000)
```

Table 36-82 Model Details View for Binning

Column Name	Description
`PARTITION_NAME`	Partition name in a partitioned model
`ATTRIBUTE_NAME`	Specifies the attribute name
`ATTRIBUTE_SUBNAME`	Specifies the attribute subname
`BIN_ID`	Bin ID (or bin identifier)
`LOWER_BIN_BOUNDARY`	Numeric lower bin boundary
`UPPER_BIN_BOUNDARY`	Numeric upper bin boundary
`ATTRIBUTE_VALUE`	Categorical value

36.8.25 Model Detail Views for Global Information

Model detail views for global information contain information about global statistics, alerts, and computed settings.

The Global Name-Value Pairs view (`DM$VG`*model_name*) describes global statistics related to the model build. Examples include the number of rows used in the build, the convergence status, and the model quality metrics. The view has the following columns:

```
Name                        Type
--------------------        --------------------
PARTITION_NAME              VARCHAR2(128)
```

```
NAME                       VARCHAR2(30)
NUMERIC_VALUE              NUMBER
STRING_VALUE               VARCHAR2(4000)
```

Table 36-83   Global Name-Value Pairs View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
NAME	Name of the statistic
NUMERIC_VALUE	Numeric value of the statistic
STRING_VALUE	Categorical value of the statistic

The Model Build Alerts view (DM$VW*model_name*) lists alerts issued during the model build. The view has the following columns:

```
Name                          Type
-------------------           ----------------------
PARTITION_NAME                VARCHAR2(128)
ERROR_NUMBER                  BINARY_DOUBLE
ERROR_TEXT                    VARCHAR2(4000)
```

Table 36-84   Model Build Alerts View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ERROR_NUMBER	Error number (valid when event is Error)
ERROR_TEXT	Error message

The Computed Settings view (DM$VS*model_name*) lists the algorithm computed settings. The view has the following columns:

```
Name                           Type
-----------------              --------------------
PARTITION_NAME                 VARCHAR2(128)
SETTING_NAME                   VARCHAR2(30)
SETTING_VALUE                  VARCHAR2(4000)
```

Table 36-85   Computed Settings View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
SETTING_NAME	Name of the setting
SETTING_VALUE	Value of the setting

36.8.26 Model Detail Views for Normalization and Missing Value Handling

The Normalization and Missing Value Handling view `DM$VN` describes the normalization parameters used in Automatic Data Preparation (ADP) and the missing value replacement when a `NULL` value is encountered. Missing value replacement applies only to the twodimensional columns and does not apply to the nested columns.

The view has the following columns:

```
Name                          Type
----------------------        ----------------------
PARTITION_NAME                VARCHAR2(128)
ATTRIBUTE_NAME                VARCHAR2(128)
ATTRIBUTE_SUBNAME             VARCHAR2(4000)
NUMERIC_MISSING_VALUE         BINARY_DOUBLE
CATEGORICAL_MISSING_VALUE     VARCHAR2(4000)
NORMALIZATION_SHIFT           BINARY_DOUBLE
NORMALIZATION_SCALE           BINARY_DOUBLE
```

Table 36-86 Normalization and Missing Value Handling View

Column Name	Description
`PARTITION_NAME`	A partition in a partitioned model
`ATTRIBUTE_NAME`	Column name
`ATTRIBUTE_SUBNAME`	Nested column subname. The value is null for non-nested columns.
`NUMERIC_MISSING_VALUE`	Numeric missing value replacement
`CATEGORICAL_MISSING_VALUE`	Categorical missing value replacement
`NORMALIZATION_SHIFT`	Normalization shift value
`NORMALIZATION_SCALE`	Normalization scale value

36.8.27 Model Detail Views for ONNX Models

You can view the details of an embedding model using the model detail views. The names of the views begin with DM$V.

This section lists the model detail views for embedding models.

36.8.27.1 DM$VJ Model Detail View

The `DM$VJ`<*model-name*> returns a single row containing a JSON object in one column that contains user-specified metadata of the model.

The view has the following columns:

```
Name                       Null?    Type
------------------------ -------- ----------------------------
 METADATA                           CLOB
```

Column Name	Description
METADATA	It is a CLOB containing the user-specified metadata of the embedding model in JSON format.

The following table describes the output of the DM$VJ<*modle_name*> view of an embedding model.

Name	Value
METADATA	The JSON that was specified to the IMPORT_ONNX_MODEL call for importing the model.

The following example displays the output of an embedding model. The name of the model is *doc_model*:

```
SQL> select * from DM$VJdoc_model;

METADATA
------------------------------------------------------------------------------
---
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
["DATA"]}}
```

36.8.27.2 DM$VM Model Detail View

The DM$VM<*model-name*> view reports information extracted from the metadata of the imported ONNX model and its input or output tensors.

The view has the following columns:

```
Name                                Type
---------------------------------- ---------------------------
NAME                             VARCHAR2(4000)
VALUE                               VARCHAR2(4000)
```

Table 36-87

Column Name	Description
NAME	The name of the metadata extracted from the ONNX model.
VALUE	Indicates a value for the metadata name

The following table describes the output of the DM$VM<*model_name*> view of an embedding model.

Name	Value
Producer Name	Name of the tools that generated the ONNX files
Graph Name	Name of the ONNX graph
Graph Description	Description given to the model
Version	Version of the model
Input	Describes the model input mapping
Output	Reports the vector information with dimension and value type

The following example displays the output of an embedding model. The name of the model is *DOC_MODEL*:

```
SQL> select * from DM$VMdoc_model;

NAME                                    VALUE
----------------------------------------
----------------------------------------
Producer Name                           onnx.compose.merge_models
Graph Name                              g_8_main_graph_main_graph
Graph Description                       Graph combining
g_8_main_graph and main_
                                        graph
                                        g_8_main_graph



                                        main_graph


Version                                 1
Input[0]                                input:string[1]
Output[0]                               embedding:float32[?,384]

6 rows selected.
```

Related Topics

- https://github.com/onnx/onnx/blob/main/docs/IR.md

36.8.27.3 DM$VP Model Detail View

The DM$VP<*model-name*> view displays information extracted from parsing the JSON metadata. The view presents the JSON metadata of the model, including both explicitly declared properties and system-assigned default values for undeclared ones.

The reported properties are specific to the machine learning model and match the mandatory and optional fields of the JSON metadata.

The view has the following columns:

```
Name                                Type
---------------------------------- ---------------------------
NAME                             VARCHAR2(4000)
VALUE                               VARCHAR2(4000)
```

Column Name	Description
NAME	Displays the JSON parameters
VALUE	Indicates the value corresponding to the JSON parameter name value pair

Note that this information is already available in the `ALL_MINING_MODEL_ATTRIBUTES` view. The following example displays all the columns available to you in the `DM$VP`*`doc_model`* view of an embedding model. In this example, *doc_model* is the name of the model.

```
SQL> select * from DM$VPdoc_model;

NAME                                      VALUE
----------------------------------------
----------------------------------------
batching                                  False
embeddingOutput                           embedding
```

37

Scoring and Deployment

Explains the scoring and deployment features of Oracle Machine Learning for SQL.

37.1 About Scoring and Deployment

Scoring is the application of models to new data. In Oracle Machine Learning for SQL, scoring is performed by SQL language functions.

Predictive functions perform classification, regression, or anomaly detection. Clustering functions assign rows to clusters. Feature extraction functions transform the input data to a set of higher order predictors. A scoring procedure is also available in the `DBMS_DATA_MINING` PL/SQL package.

Deployment refers to the use of models in a target environment. Once the models have been built, the challenges come in deploying them to obtain the best results, and in maintaining them within a production environment. Deployment can be any of the following:

- Scoring data either for batch or real-time results. Scores can include predictions, probabilities, rules, and other statistics.
- Extracting model details to produce reports. For example: clustering rules, decision tree rules, or attribute rankings from an Attribute Importance model.
- Extending the business intelligence infrastructure of a data warehouse by incorporating machine learning results in applications or operational systems.
- Moving a model from the database where it was built to the database where it used for scoring (export/import)

OML4SQL supports all of these deployment scenarios.

> **Note:**
>
> OML4SQL scoring operations support parallel execution. When parallel execution is enabled, multiple CPU and I/O resources are applied to the execution of a single database operation.
>
> Parallel execution offers significant performance improvements, especially for operations that involve complex queries and large databases typically associated with decision support systems (DSS) and data warehouses.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Machine Learning for SQL Concepts*
- Export and Import Oracle Machine Learning for SQL Models
  You can export machine learning models to move models to a different Oracle Database instance, such as from a development database to a production database.

37.2 Use the Oracle Machine Learning for SQL Functions

Some of the benefits of using SQL functions for Oracle Machine Learning for SQL are listed.

The OML4SQL functions provide the following benefits:

- Models can be easily deployed within the context of existing SQL applications.
- Scoring operations take advantage of existing query execution functionality. This provides performance benefits.
- Scoring results are pipelined, enabling the rows to be processed without requiring materialization.

The machine learning functions produce a score for each row in the selection. The functions can apply a machine learning model schema object to compute the score, or they can score dynamically without a pre-defined model, as described in "Dynamic Scoring".

Related Topics

- Dynamic Scoring
  You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.
- Scoring Requirements
  Learn how scoring is done in Oracle Machine Learning for SQL.
- Oracle Machine Learning for SQL Scoring Functions
  Use OML4SQL functions score data. Functions can apply a machine learning model schema object to data or dynamically mine it with an analytic clause. SQL functions exist for all OML4SQL scoring algorithms.
- *Oracle Database SQL Language Reference*

37.2.1 Choose the Predictors

You can select different attributes as predictors in a `PREDICTION` function through a `USING` clause.

The OML4SQL functions support a `USING` clause that specifies which attributes to use for scoring. You can specify some or all of the attributes in the selection and you can specify expressions. The following examples all use the `PREDICTION` function to find the customers who are likely to use an affinity card, but each example uses a different set of predictors.

When predictor values are not in the training data, the models score categorical values that were not in the training data without error. A score is produced using the remaining predictors. This enables batch scoring that does not fail because of a single record with an invalid value. Also, in some algorithms, like k-Means or Gaussian SVM, a new value can change the prediction in a meaningful way, such as resulting in larger distances with the unknown value. Furthermore, additional columns that were not present for building may be present in the table or view provided for scoring, and only the columns matching the model signature are used. Also, scoring may be performed with fewer predictors than are listed in the model signature.

In the case of partitioned models, a `NULL` score is produced if the partition value is invalid. If the partition column value is omitted, an error message is returned.

The query in Example 37-1 uses all the predictors.

The query in Example 37-2 uses only gender, marital status, occupation, and income as predictors.

The query in Example 37-3 uses three attributes and an expression as predictors. The prediction is based on gender, marital status, occupation, and the assumption that all customers are in the highest income bracket.

Example 37-1 Using All Predictors

The dt_sh_clas_sample model is created by the `oml4sql-classification-decision-tree.sql` example.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
     FROM mining_data_apply_v
     WHERE PREDICTION(dt_sh_clas_sample USING *) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

The output is follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         25         38
M        213         43
```

Example 37-2 Using Some Predictors

```
 SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
     FROM mining_data_apply_v
     WHERE PREDICTION(dt_sh_clas_sample USING
```

```
                         cust_gender,cust_marital_status,
                         occupation, cust_income_level) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

The output is as follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         30         38
M        186         43
```

Example 37-3   Using Some Predictors and an Expression

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
     FROM mining_data_apply_v
     WHERE PREDICTION(dt_sh_clas_sample USING
                     cust_gender, cust_marital_status, occupation,
                    'L: 300,000 and above' AS cust_income_level) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

The output is follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         30         38
M        186         43
```

37.2.2 Single-Record Scoring

You can score a single record which produces 0 and 1 to predict customers who are unlikely or likely to use an affinity card.

The Oracle Machine Learning for SQL functions can produce a score for a single record, as shown in Example 37-4 and Example 37-5.

Example 37-4 returns a prediction for customer 102001 by applying the classification model NB_SH_Clas_sample. The resulting score is 0, meaning that this customer is unlikely to use an affinity card. The NB_SH_Clas_Sample model is created by the `oml4sql-classification-naive-bayes.sql` example.

Example 37-5 returns a prediction for 'Affinity card is great' as the comments attribute by applying the text machine learning model T_SVM_Clas_sample. The resulting score is 1, meaning that this customer is likely to use an affinity card. The T_SVM_Clas_sample model is created by the `oml4sql-classification-text-analysis-svm.sql` example.

Example 37-4   Scoring a Single Customer or a Single Text Expression

```
SELECT PREDICTION (NB_SH_Clas_Sample USING *)
    FROM sh.customers where cust_id = 102001;
```

The output is as follows:

```
PREDICTION(NB_SH_CLAS_SAMPLEUSING*)
-----------------------------------
                                  0
```

Example 37-5   Scoring a Single Text Expression

```
SELECT
  PREDICTION(T_SVM_Clas_sample USING 'Affinity card is great' AS comments)
FROM DUAL;
```

The output is as follows:

```
PREDICTION(T_SVM_CLAS_SAMPLEUSING'AFFINITYCARDISGREAT'ASCOMMENTS)
-----------------------------------------------------------------
                                                                1
```

37.3 Prediction Details

Prediction details are XML strings that provide information about the score.

Details are available for all types of scoring: clustering, feature extraction, classification, regression, and anomaly detection. Details are available whether scoring is dynamic or the result of model apply.

The details functions, `CLUSTER_DETAILS`, `FEATURE_DETAILS`, and `PREDICTION_DETAILS` return the actual value of attributes used for scoring and the relative importance of the attributes in determining the score. By default, the functions return the five most important attributes in descending order of importance.

37.3.1 Cluster Details

Shows an example of the `CLUSTER_DETAILS` function.

For the most likely cluster assignments of customer 100955 (probability of assignment > 20%), the query in the following example produces the five attributes that have the most impact for each of the likely clusters. The clustering functions apply an Expectation Maximization model named em_sh_clus_sample to the data selected from `mining_data_apply_v`. The "5" specified in `CLUSTER_DETAILS` is not required, because five attributes are returned by default. The em_sh_clus_sample model is created by the `oml4sql-clustering-expectation-maximization.sql` example.

Example 37-6   Cluster Details

```
SELECT S.cluster_id, probability prob,
           CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 USING T.*) det
    FROM
      (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
        FROM mining_data_apply_v v
       WHERE cust_id = 100955) T,
```

```
   TABLE(T.pset) S
 ORDER BY 2 DESC;
```

The output is as follows:

```
CLUSTER_ID  PROB DET
---------- -----
----------------------------------------------------------------------
-----
        14 .6761 <Details algorithm="Expectation Maximization"
cluster="14">
                 <Attribute name="AGE" actualValue="51" weight=".676"
rank="1"/>
                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1" weight=".557" rank="2"/>
                 <Attribute name="FLAT_PANEL_MONITOR" actualValue="0"
weight=".412" rank="3"/>
                 <Attribute name="Y_BOX_GAMES" actualValue="0"
weight=".171" rank="4"/>
                 <Attribute
name="BOOKKEEPING_APPLICATION"actualValue="1" weight="-.003"
                  rank="5"/>
                 </Details>

         3 .3227 <Details algorithm="Expectation Maximization"
cluster="3">
                 <Attribute name="YRS_RESIDENCE" actualValue="3"
weight=".323" rank="1"/>
                 <Attribute name="BULK_PACK_DISKETTES" actualValue="1"
weight=".265" rank="2"/>
                 <Attribute name="EDUCATION" actualValue="HS-grad"
weight=".172" rank="3"/>
                 <Attribute name="AFFINITY_CARD" actualValue="0"
weight=".125" rank="4"/>
                 <Attribute name="OCCUPATION" actualValue="Crafts"
weight=".055" rank="5"/>
                 </Details>
```

37.3.2 Feature Details

Shows an example of the `FEATURE_DETAILS` function.

The query in the following example returns the three attributes that have the greatest impact on the top Principal Components Analysis (PCA) projection for customer 101501. The `FEATURE_DETAILS` function applies a Singular Value Decomposition (SVD) model named svd_sh_sample to the data selected from the svd_sh_sample_build_num table. The table and model are created by the `oml4sql-singular-value-decomposition.sql` example.

Example 37-7   Feature Details

```
SELECT FEATURE_DETAILS(svd_sh_sample, 1, 3 USING *) proj1det
  FROM svd_sh_sample_build_num
  WHERE CUST_ID = 101501;
```

The output is as follows:

```
PROJ1DET
--------------------------------------------------------------------------------
---
<Details algorithm="Singular Value Decomposition" feature="1">
<Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".352"
rank="1"/>
<Attribute name="Y_BOX_GAMES" actualValue="0" weight=".249" rank="2"/>
<Attribute name="AGE" actualValue="41" weight=".063" rank="3"/>
</Details>
```

37.3.3 Prediction Details

Shows an examples of `PREDICTION_DETAILS` function.

The query in the following example returns the attributes that are most important in predicting the age of customer 100010. The prediction functions apply a Generalized Linear Model regression model named GLMR_SH_Regr_sample to the data selected from `mining_data_apply_v`. The GLMR_SH_Regr_sample model is created by the `oml4sql-regression-glm.sql` example.

Example 37-8 Prediction Details for Regression

```
SELECT cust_id,
      PREDICTION(GLMR_SH_Regr_sample USING *) pr,
      PREDICTION_DETAILS(GLMR_SH_Regr_sample USING *) pd
  FROM mining_data_apply_v
  WHERE CUST_ID = 100010;
```

The output is as follows:

```
CUST_ID    PR PD
------- ----- -----------
 100010 25.45 <Details algorithm="Generalized Linear Model">
              <Attribute name="FLAT_PANEL_MONITOR" actualValue="1"
weight=".025" rank="1"/>
              <Attribute name="OCCUPATION" actualValue="Crafts"
weight=".019" rank="2"/>
              <Attribute name="AFFINITY_CARD" actualValue="0" weight=".01"
rank="3"/>
              <Attribute name="OS_DOC_SET_KANJI" actualValue="0" weight="0"
rank="4"/>
              <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight="-.004" rank="5"/>
              </Details>
```

The query in the following example returns the customers who work in Tech Support and are likely to use an affinity card (with more than 85% probability). The prediction functions apply an Support Vector Machine (SVM) classification model named svmc_sh_clas_sample. to the data selected from `mining_data_apply_v`. The query includes the prediction details, which show that education is the most important predictor. The svmc_sh_clas_sample model is created by the `oml4sql-classification-svm.sql` example.

Example 37-9 Prediction Details for Classification

```
SELECT cust_id, PREDICTION_DETAILS(svmc_sh_clas_sample, 1 USING *) PD
      FROM mining_data_apply_v
  WHERE PREDICTION_PROBABILITY(svmc_sh_clas_sample, 1 USING *) > 0.85
  AND occupation = 'TechSup'
  ORDER BY cust_id;
```

The output is as follows:

```
CUST_ID PD
-------
-----------------------------------------------------------------------
----------------
 100029 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A"
weight=".199" rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="I: 170\,000 -
189\,999" weight=".044"
         rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".028" rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1"
weight=".024" rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".022" rank="5"/>
        </Details>

 100378 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A" weight=".21"
rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="B: 30\,000 -
49\,999" weight=".047"
         rank="2"/>
        <Attribute name="FLAT_PANEL_MONITOR" actualValue="0"
weight=".043" rank="3"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".03" rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".023" rank="5"/>
        </Details>

 100508 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Bach." weight=".19"
rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="L: 300\,000
and above" weight=".046"
         rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".031" rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1"
```

```
weight=".026" rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".024" rank="5"/>
        </Details>

 100980 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A" weight=".19"
rank="1"/>
        <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".038"
rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".026"
rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".022"
rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".02" rank="5"/>
        </Details>
```

The query in the following example returns the two customers that differ the most from the rest of the customers. The prediction functions apply an anomaly detection model named SVMO_SH_Clas_sample to the data selected from `mining_data_apply_v`. anomaly detection uses a one-class SVM classifier. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

Example 37-10 Prediction Details for Anomaly Detection

```
SELECT cust_id, pd FROM
  (SELECT cust_id,
        PREDICTION_DETAILS(SVMO_SH_Clas_sample, 0 USING *) pd,
        RANK() OVER (ORDER BY prediction_probability(
              SVMO_SH_Clas_sample, 0 USING *) DESC, cust_id) rnk
  FROM mining_data_one_class_v)
  WHERE rnk <= 2
  ORDER BY rnk;
```

The output is as follows:

```
  CUST_ID PD
----------
-----------------------------------------------------------------------------
------
    102366 <Details algorithm="Support Vector Machines" class="0">
           <Attribute name="COUNTRY_NAME" actualValue="United Kingdom"
weight=".078" rank="1"/>
           <Attribute name="CUST_MARITAL_STATUS" actualValue="Divorc."
weight=".027" rank="2"/>
           <Attribute name="CUST_GENDER" actualValue="F" weight=".01"
rank="3"/>
           <Attribute name="HOUSEHOLD_SIZE" actualValue="9+" weight=".009"
rank="4"/>
           <Attribute name="AGE" actualValue="28" weight=".006" rank="5"/>
           </Details>

    101790 <Details algorithm="Support Vector Machines" class="0">
           <Attribute name="COUNTRY_NAME" actualValue="Canada" weight=".068"
```

```
rank="1"/>
           <Attribute name="HOUSEHOLD_SIZE" actualValue="4-5"
weight=".018" rank="2"/>
           <Attribute name="EDUCATION" actualValue="7th-8th"
weight=".015" rank="3"/>
           <Attribute name="CUST_GENDER" actualValue="F" weight=".013"
rank="4"/>
           <Attribute name="AGE" actualValue="38" weight=".001"
rank="5"/>
           </Details>
```

37.3.4 GROUPING Hint

OML4SQL functions include `PREDICTION*`, `CLUSTER*`, `FEATURE*`, and `ORA_DM_*`. The `GROUPING` hint is an optional hint that applies to machine learning scoring functions when scoring partitioned models.

This hint results in partitioning the input data set into distinct data slices so that each partition is scored in its entirety before advancing to the next partition. However, parallelism by partition is still available. Data slices are determined by the partitioning key columns used when the model was built. This method can be used with any machine learning function against a partitioned model. The hint may yield a query performance gain when scoring large data that is associated with many partitions but may negatively impact performance when scoring large data with few partitions on large systems. Typically, there is no performance gain if you use the hint for single row queries.

Enhanced PREDICTION Function Command Format

```
<prediction function> ::=
    PREDICTION <left paren> /*+ GROUPING */ <prediction model>
        [ <comma> <class value> [ <comma> <top N> ] ]
        USING <machine learning attribute list> <right paren>
```

The syntax for only the `PREDICTION` function is given but it is applicable to any machine learning function in which `PREDICTION`, `CLUSTERING`, and `FEATURE_EXTRACTION` scoring functions occur.

Example 37-11 Example

```
SELECT PREDICTION(/*+ GROUPING */my_model USING *) pred FROM <input
table>;
```

Related Topics

- *Oracle Database SQL Language Reference*

37.4 Real-Time Scoring

You can perform real-time scoring by running a SQL query. An example shows a real-time query using `PREDICTION_PROBABILITY` function. Based on the result, a customer representative can offer a value card to the customer.

Oracle Machine Learning for SQL functions enable prediction, clustering, and feature extraction analysis to be easily integrated into live production and operational systems. Because machine learning results are returned within SQL queries, machine learning can occur in real time.

With real-time scoring, point-of-sales database transactions can be mined. Predictions and rule sets can be generated to help front-line workers make better analytical decisions. Real-time scoring enables fraud detection, identification of potential liabilities, and recognition of better marketing and selling opportunities.

The query in the following example uses a Decision Tree model named `dt_sh_clas_sample` to predict the probability that customer 101488 uses an affinity card. A customer representative can retrieve this information in real time when talking to this customer on the phone. Based on the query result, the representative can offer an extra-value card, since there is a 73% chance that the customer uses a card. The model is created by the `oml4sql-classification-decision-tree.sql` example.

Example 37-12 Real-Time Query with Prediction Probability

```
SELECT PREDICTION_PROBABILITY(dt_sh_clas_sample, 1 USING *) cust_card_prob
       FROM mining_data_apply_v
       WHERE cust_id = 101488;
```

The output is as follows:

```
CUST_CARD_PROB
--------------
        .72764
```

37.5 Dynamic Scoring

You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.

The Oracle Machine Learning for SQL functions operate in two modes: by applying a predefined model, or by executing an analytic clause. If you supply an analytic clause instead of a model name, the function builds one or more transient models and uses them to score the data.

The ability to score data dynamically without a predefined model extends the application of basic embedded machine learning techniques into environments where models are not available. Dynamic scoring, however, has limitations. The transient models created during dynamic scoring are not available for inspection or fine tuning. Applications that require model inspection, the correlation of scoring results with the model, special algorithm settings, or multiple scoring queries that use the same model, require a predefined model.

The following example shows a dynamic scoring query. The example identifies the rows in the input data that contain unusual customer age values.

Example 37-13 Dynamic Prediction

```
SELECT cust_id, age, pred_age, age-pred_age age_diff, pred_det FROM
 (SELECT cust_id, age, pred_age, pred_det,
    RANK() OVER (ORDER BY ABS(age-pred_age) DESC) rnk FROM
    (SELECT cust_id, age,
         PREDICTION(FOR age USING *) OVER () pred_age,
```

```
        PREDICTION_DETAILS(FOR age ABS USING *) OVER () pred_det
  FROM mining_data_apply_v))
WHERE rnk <= 5;
```

The output is follows:

```
CUST_ID  AGE   PRED_AGE AGE_DIFF PRED_DET
------- ---- ---------- --------
--------------------------------------------------------------
 100910   80 40.6686505    39.33 <Details algorithm="Support Vector
Machines">
                                 <Attribute
name="HOME_THEATER_PACKAGE" actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0"
                                  weight=".059" rank="2"/>
                                 <Attribute name="AFFINITY_CARD"
actualValue="0"
                                  weight=".059" rank="3"/>
                                 <Attribute name="FLAT_PANEL_MONITOR"
actualValue="1"
                                  weight=".059" rank="4"/>
                                 <Attribute name="YRS_RESIDENCE"
actualValue="4"
                                  weight=".059" rank="5"/>
                                  </Details>

 101285   79 42.1753571    36.82 <Details algorithm="Support Vector
Machines">
                                 <Attribute
name="HOME_THEATER_PACKAGE" actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="HOUSEHOLD_SIZE"
actualValue="2" weight=".059"
                                  rank="2"/>
                                 <Attribute name="CUST_MARITAL_STATUS"
actualValue="Mabsent"
                                  weight=".059" rank="3"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="4"/>
                                 <Attribute name="OCCUPATION"
actualValue="Prof." weight=".059"
                                  rank="5"/>
                                 </Details>

 100694   77 41.0396722    35.96 <Details algorithm="Support Vector
Machines">
                                 <Attribute
name="HOME_THEATER_PACKAGE" actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="EDUCATION"
actualValue="&lt; Bach."
```

```
                                  weight=".059" rank="2"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="3"/>
                                 <Attribute name="CUST_ID"
actualValue="100694" weight=".059"
                                  rank="4"/>
                                 <Attribute name="COUNTRY_NAME"
actualValue="United States of
                                  America" weight=".059" rank="5"/>
                                 </Details>

 100308    81 45.3252491    35.67 <Details algorithm="Support Vector
Machines">
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="2"/>
                                 <Attribute name="HOUSEHOLD_SIZE"
actualValue="2" weight=".059"
                                  rank="3"/>
                                 <Attribute name="FLAT_PANEL_MONITOR"
actualValue="1"
                                  weight=".059" rank="4"/>
                                 <Attribute name="CUST_GENDER"
actualValue="F" weight=".059"
                                  rank="5"/>
                                 </Details>

 101256    90 54.3862214    35.61 <Details algorithm="Support Vector
Machines">
                                 <Attribute name="YRS_RESIDENCE"
actualValue="9" weight=".059"
                                  rank="1"/>
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="2"/>
                                 <Attribute name="EDUCATION"
actualValue="&lt; Bach."
                                  weight=".059" rank="3"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="4"/>
                                 <Attribute name="COUNTRY_NAME"
actualValue="United States of
                                  America" weight=".059" rank="5"/>
                                 </Details>
```

37.6 Cost-Sensitive Decision Making

Costs are user-specified numbers that bias classification. The algorithm uses positive numbers to penalize more expensive outcomes over less expensive outcomes. Higher numbers indicate higher costs.

The algorithm uses negative numbers to favor more beneficial outcomes over less beneficial outcomes. Lower negative numbers indicate higher benefits.

All classification algorithms can use costs for scoring. You can specify the costs in a cost matrix table, or you can specify the costs inline when scoring. If you specify costs inline and the model also has an associated cost matrix, only the inline costs are used. The `PREDICTION`, `PREDICTION_SET`, and `PREDICTION_COST` functions support costs.

Only the Decision Tree algorithm can use costs to bias the model build. If you want to create a Decision Tree model with costs, create a cost matrix table and provide its name in the `CLAS_COST_TABLE_NAME` setting for the model. If you specify costs when building the model, the cost matrix used to create the model is used when scoring. If you want to use a different cost matrix table for scoring, first remove the existing cost matrix table then add the new one.

A sample cost matrix table is shown in the following table. The cost matrix specifies costs for a binary target. The matrix indicates that the algorithm must treat a misclassified 0 as twice as costly as a misclassified 1.

Table 37-1 Sample Cost Matrix

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	COST
0	0	0
0	1	2
1	0	1
1	1	0

Example 37-14 Sample Queries With Costs

The table `nbmodel_costs` contains the cost matrix described in Table 37-1.

```
SELECT * from nbmodel_costs;
```

The output is as follows:

```
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1          2
                  1                      0          1
                  1                      1          0
```

The following statement associates the cost matrix with a Naive Bayes model called `nbmodel`.

```
BEGIN
  dbms_data_mining.add_cost_matrix('nbmodel', 'nbmodel_costs');
END;
/
```

The following query takes the cost matrix into account when scoring `mining_data_apply_v`. The output is restricted to those rows where a prediction of 1 is less costly then a prediction of 0.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
        FROM mining_data_apply_v
        WHERE PREDICTION (nbmodel COST MODEL
       USING cust_marital_status, education, household_size) = 1
        GROUP BY cust_gender
        ORDER BY cust_gender;
```

The output is as follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         25         38
M        208         43
```

You can specify costs inline when you invoke the scoring function. If you specify costs inline and the model also has an associated cost matrix, only the inline costs are used. The same query is shown below with different costs specified inline. Instead of the "2" shown in the cost matrix table (Table 37-1), "10" is specified in the inline costs.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
      FROM mining_data_apply_v
      WHERE PREDICTION (nbmodel
                COST (0,1) values ((0, 10),
                           (1, 0))
                USING cust_marital_status, education, household_size) = 1
      GROUP BY cust_gender
      ORDER BY cust_gender;
```

The output is as follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         74         39
M        581         43
```

The same query based on probability instead of costs is shown below.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
        FROM mining_data_apply_v
        WHERE PREDICTION (nbmodel
           USING cust_marital_status, education, household_size) = 1
        GROUP BY cust_gender
        ORDER BY cust_gender;
```

The output is as follows:

```
C        CNT    AVG_AGE
- ---------- ----------
F         73         39
M        577         44
```

Related Topics

- Example 33-1

37.7 DBMS_DATA_MINING.APPLY

The `APPLY` procedure in `DBMS_DATA_MINING` is a batch apply operation that writes the results of scoring directly to a table.

The columns in the table are machine learning function-dependent.

Scoring with `APPLY` generates the same results as scoring with the SQL scoring functions. Classification produces a prediction and a probability for each case; clustering produces a cluster ID and a probability for each case, and so on. The difference lies in the way that scoring results are captured and the mechanisms that can be used for retrieving them.

`APPLY` creates an output table with the columns shown in the following table:

Table 37-2 APPLY Output Table

Machine Learning Technique	Output Columns

classification	`CASE_ID`
	`PREDICTION`
	`PROBABILITY`

regression	`CASE_ID` `PREDICTION`

anomaly detection	`CASE_ID`
	`PREDICTION`
	`PROBABILITY`

clustering	`CASE_ID`
	`CLUSTER_ID`
	`PROBABILITY`

feature extraction	`CASE_ID`
	`FEATURE_ID`
	`MATCH_QUALITY`

Since `APPLY` output is stored separately from the scoring data, it must be joined to the scoring data to support queries that include the scored rows. Thus any model that is used with `APPLY` must have a case ID.

A case ID is not required for models that is applied with SQL scoring functions. Likewise, storage and joins are not required, since scoring results are generated and consumed in real time within a SQL query.

The following example illustrates anomaly detection with `APPLY`. The query of the `APPLY` output table returns the ten first customers in the table. Each has a a probability for being typical (1) and a probability for being anomalous (0). The SVMO_SH_Clas_sample model is created by the `oml4sql-anomaly-detection-1class-svm.sql` example.

Example 37-15   Anomaly Detection with DBMS_DATA_MINING.APPLY

```
EXEC dbms_data_mining.apply
        ('SVMO_SH_Clas_sample','svmo_sh_sample_prepared',
         'cust_id', 'one_class_output');

SELECT * from one_class_output where rownum < 11;
```

The output is as follows:

```
   CUST_ID PREDICTION PROBABILITY
---------- ---------- -----------
    101798          1  .567389309
    101798          0  .432610691
    102276          1  .564922469
    102276          0  .435077531
    102404          1   .51213544
    102404          0   .48786456
    101891          1  .563474346
    101891          0  .436525654
    102815          0  .500663683
    102815          1  .499336317
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

38 Machine Learning Operations on Unstructured Text

Explains how to use Oracle Machine Learning for SQL to operate on unstructured text.

- About Unstructured Text
- About Machine Learning and Oracle Text
- Create a Model that Includes Machine Learning Operations on Text
- Create a Text Policy
- Configure a Text Attribute

38.1 About Unstructured Text

Unstructured text may contain important information that is critical to the success of a business.

Machine learning algorithms act on data that is numerical or categorical. Numerical data is ordered. It is stored in columns that have a numeric data type, such as `NUMBER` or `FLOAT`. Categorical data is identified by category or classification. It is stored in columns that have a character data type, such as `VARCHAR2` or `CHAR`.

Unstructured text data is neither numerical nor categorical. Unstructured text includes items such as web pages, document libraries, Power Point presentations, product specifications, emails, comment fields in reports, and call center notes. It has been said that unstructured text accounts for more than three quarters of all enterprise data. Extracting meaningful information from unstructured text can be critical to the success of a business.

38.2 About Machine Learning and Oracle Text

Understand machine learning operations on text and Oracle Text.

Machine learning operations on text is the process of applying machine learning techniques to text terms, also called text features or tokens. Text terms are words or groups of words that have been extracted from text documents and assigned numeric weights. Text terms are the fundamental unit of text that can be manipulated and analyzed.

Oracle Text is an Oracle Database technology that provides term extraction, word and theme searching, and other utilities for querying text. When columns of text are present in the training data, Oracle Machine Learning for SQL uses Oracle Text utilities and term weighting strategies to transform the text for machine learning operations. OML4SQL passes configuration information supplied by you to Oracle Text and uses the results in the model creation process.

Related Topics

- *Oracle Text Application Developer's Guide*

38.3 Model Detail Views for Text Features

The model details view for text features is `DM$VX`*model_name*.

The text feature view `DM$VX`*model_name* describes the extracted text features if there are text attributes present. The view has the following schema:

```
Name                                    Type
 --------------                    ---------------------
 PARTITION_NAME                         VARCHAR2(128)
 COLUMN_NAME                            VARCHAR2(128)
 TOKEN                                  VARCHAR2(4000)
 DOCUMENT_FREQUENCY                     NUMBER
```

Table 38-1 Text Feature View for Extracted Text Features

Column Name	Description
`PARTITION_NAME`	A partition in a partitioned model to retrieve details
`COLUMN_NAME`	Name of the identifier column
`TOKEN`	Text token which is usually a word or stemmed word
`DOCUMENT_FREQUENCY`	A measure of token frequency in the entire training set

38.4 Create a Model that Includes Machine Learning Operations on Text

Create a model and specify the settings to perform machine learning operations on text.

Oracle Machine Learning for SQL supports unstructured text within columns of `VARCHAR2`, `CHAR`, `CLOB`, `BLOB`, and `BFILE`, as described in the following table:

Table 38-2 Column Data Types That May Contain Unstructured Text

Data Type	Description
`BFILE` and `BLOB`	Oracle Machine Learning for SQL interprets `BLOB` and `BFILE` as text *only if* you identify the columns as text when you create the model. If you do not identify the columns as text, then `CREATE_MODEL` returns an error.
`CLOB`	OML4SQL interprets `CLOB` as text.
`CHAR`	OML4SQL interprets `CHAR` as categorical by default. You can identify columns of `CHAR` as text when you create the model.
`VARCHAR2`	OML4SQL interprets `VARCHAR2` with data length > 4000 as text. OML4SQL interprets `VARCHAR2` with data length <= 4000 as categorical by default. You can identify these columns as text when you create the model.

> **Note:**
>
> Text is not supported in nested columns or as a target in supervised machine learning.

The settings described in the following table control the term extraction process for text attributes in a model. Instructions for specifying model settings are in "Specifying Model Settings".

Table 38-3 Model Settings for Text

Setting Name	Data Type	Setting Value	Description
ODMS_TEXT_POLICY_NAME	VARCHAR2(4000)	Name of an Oracle Text policy object created with CTX_DDL.CREATE_POLICY	Affects how individual tokens are extracted from unstructured text.
ODMS_TEXT_MAX_FEATURES	INTEGER	1 <= *value* <= 100000	Maximum number of features to use from the document set (across all documents of each text column) passed to CREATE_MODEL. Default is 3000.

A model can include one or more text attributes. A model with text attributes can also include categorical and numerical attributes.

To create a model that includes text attributes:

1. Create an Oracle Text policy object.
2. Specify the model configuration settings that are described in "Table 38-3".
3. Specify which columns must be treated as text and, optionally, provide text transformation instructions for individual attributes.
4. Pass the model settings and text transformation instructions to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

> **Note:**
>
> All algorithms except O-Cluster can support columns of unstructured text.
>
> The use of unstructured text is not recommended for association rules (Apriori).

In the following example, an SVM model is used to predict customers that are most likely to be positive responders to an Affinity Card loyalty program. The data comes with a text column that contains user generated comments. By creating an Oracle Text policy and specifying model settings, the algorithm automatically uses the text column and builds the model on both the structured data and unstructured text.

This example uses a view called `mining_data` which is created from `SH.SALES` table. A training data set called `mining_train_text` is also created.

The following queries show you how to create an Oracle Text policy followed by building a model using `CREATE_MODEL2` procedure.

```
%script

BEGIN

EXECUTE ctx_ddl.create_policy('dmdemo_svm_policy');
```

The output is:

```


PL/SQL procedure successfully completed.

---------------------------

PL/SQL procedure successfully completed.


%script

BEGIN DBMS_DATA_MINING.DROP_MODEL('T_SVM_Clas_sample');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
    xformlist dbms_data_mining_transform.TRANSFORM_LIST;

BEGIN

    v_setlst(dbms_data_mining.algo_name) :=
dbms_data_mining.algo_support_vector_machines;
    v_setlst(dbms_data_mining.prep_auto) :=  dbms_data_mining.prep_auto_on;
    v_setlst(dbms_data_mining.svms_kernel_function) :=
dbms_data_mining.svms_linear;
    v_setlst(dbms_data_mining.svms_complexity_factor) := '100';
    v_setlst(dbms_data_mining.odms_text_policy_name) := 'DMDEMO_SVM_POLICY';

    v_setlst(dbms_data_mining.svms_solver) :=  dbms_data_mining.svms_solver_sgd;
    dbms_data_mining_transform.SET_TRANSFORM(
        xformlist, 'comments', null, 'comments', null, 'TEXT');
    DBMS_DATA_MINING.CREATE_MODEL2(
        model_name          => 'T_SVM_Clas_sample',
        mining_function     => dbms_data_mining.classification,
        data_query          => 'select * from mining_train_text',
        set_list            => v_setlst,
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        xform_list => xformlist);
END;
/
```

The output is:

```



PL/SQL procedure successfully completed.

---------------------------
```

```
PL/SQL procedure successfully completed.

---------------------------
```

Related Topics

- Specify Model Settings
  You can configure your model by specifying model settings.
- Create a Text Policy
  An Oracle Text policy specifies how text content must be interpreted. You can provide a text policy to govern a model, an attribute, or both the model and individual attributes.
- Configure a Text Attribute
  Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.
- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

38.5 Create a Text Policy

An Oracle Text policy specifies how text content must be interpreted. You can provide a text policy to govern a model, an attribute, or both the model and individual attributes.

If a model-specific policy is present and one or more attributes have their own policies, Oracle Machine Learning for SQL uses the attribute policies for the specified attributes and the model-specific policy for the other attributes.

The `CTX_DDL.CREATE_POLICY` procedure creates a text policy.

```
CTX_DDL.CREATE_POLICY(
         policy_name    IN VARCHAR2,
                         filter         IN VARCHAR2 DEFAULT NULL,
                         section_group  IN VARCHAR2 DEFAULT NULL,
                         lexer          IN VARCHAR2 DEFAULT NULL,
                         stoplist       IN VARCHAR2 DEFAULT NULL,
                         wordlist       IN VARCHAR2 DEFAULT NULL);
```

The parameters of `CTX_DDL.CREATE_POLICY` are described in the following table.

Table 38-4 CTX_DDL.CREATE_POLICY Procedure Parameters

Parameter Name	Description
`policy_name`	Name of the new policy object. Oracle Text policies and text indexes share the same namespace.
`filter`	Specifies how the documents must be converted to plain text for indexing. Examples are: `CHARSET_FILTER` for character sets and `NULL_FILTER` for plain text, HTML and XML. For `filter` values, see "Filter Types" in *Oracle Text Reference*.

Table 38-4 (Cont.) CTX_DDL.CREATE_POLICY Procedure Parameters

Parameter Name	Description
`section_group`	Identifies sections within the documents. For example, `HTML_SECTION_GROUP` defines sections in HTML documents. For `section_group` values, see "Section Group Types" in *Oracle Text Reference*. Note: You can specify any section group that is supported by `CONTEXT` indexes.
`lexer`	Identifies the language that is being indexed. For example, `BASIC_LEXER` is the lexer for extracting terms from text in languages that use white space delimited words (such as English and most western European languages). For `lexer` values, see "Lexer Types" in *Oracle Text Reference*.
`stoplist`	Specifies words and themes to exclude from term extraction. For example, the word "the" is typically in the stoplist for English language documents. The system-supplied stoplist is used by default. See "Stoplists" in *Oracle Text Reference*.
`wordlist`	Specifies how stems and fuzzy queries must be expanded. A stem defines a root form of a word so that different grammatical forms have a single representation. A fuzzy query includes common misspellings in the representation of a word. See "`BASIC_WORDLIST`" in *Oracle Text Reference*.

Related Topics

- *Oracle Text Reference*

38.6 Configure a Text Attribute

Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.

As shown in Table 38-2, you can identify columns of `CHAR`, shorter `VARCHAR2` (<=4000), `BFILE`, and `BLOB` as text attributes. If `CHAR` and shorter `VARCHAR2` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` processes them as categorical attributes. If `BFILE` and `BLOB` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` returns an error.

To identify a column as a text attribute, supply the keyword `TEXT` in an **Attribute specification**. The attribute specification is a field (`attribute_spec`) in a transformation record (`transform_rec`). Transformation records are components of transformation lists (`xform_list`) that can be passed to `CREATE_MODEL`or `CREATE_MODEL2`.

> **Note:**
>
> An attribute specification can also include information that is not related to text. Instructions for constructing an attribute specification are in "Embedding Transformations in a Model".

You can provide transformation instructions for any text attribute by qualifying the TEXT keyword in the attribute specification with the subsettings described in the following table.

Table 38-5 Attribute-Specific Text Transformation Instructions

Subsetting Name	Description	Example
BIGRAM	A sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. Here, NORMAL tokens are mixed with their bigrams.	(TOKEN_TYPE:*BIGRAM*)
POLICY_NAME	Name of an Oracle Text policy object created with CTX_DDL.CREATE_POLICY	(POLICY_NAME:*my_policy*)
STEM_BIGRAM	Here, STEM tokens are extracted first and then stem bigrams are formed.	(TOKEN_TYPE:*STEM_BIGRAM*)
SYNONYM	Oracle Machine Learning for SQL supports synonyms. The following is an optional parameter: <thesaurus> where <thesaurus> is the name of the thesaurus defining synonyms. If SYNONYM is used without this parameter, then the default thesaurus is used.	(TOKEN_TYPE:*SYNONYM*) (TOKEN_TYPE:*SYNONYM[NAMES]*)
TOKEN_TYPE	The following values are supported: NORMAL (the default) STEM THEME See "Token Types in an Attribute Specification"	(TOKEN_TYPE:THEME)
MAX_FEATURES	Maximum number of features to use from the attribute.	(MAX_FEATURES:3000)

Note:

The TEXT keyword is only required for CLOB and longer VARCHAR2 (>4000) when you specify transformation instructions. The TEXT keyword is *always* required for CHAR, shorter VARCHAR2, BFILE, and BLOB — whether or not you specify transformation instructions.

Tip:

You can view attribute specifications in the data dictionary view ALL_MINING_MODEL_ATTRIBUTES, as shown in *Oracle Database Reference*.

Token Types in an Attribute Specification

When stems or themes are specified as the token type, the lexer preference for the text policy must support these types of tokens.

The following example adds themes and English stems to BASIC_LEXER.

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'BASIC_LEXER');
  CTX_DDL.SET_ATTRIBUTE('my_lexer', 'index_stems', 'ENGLISH');
  CTX_DDL.SET_ATTRIBUTE('my_lexer', 'index_themes', 'YES');
END;
```

Example 38-1 A Sample Attribute Specification for Text

This expression specifies that text transformation for the attribute must use the text policy named `my_policy`. The token type is `THEME`, and the maximum number of features is 3000.

```
"TEXT(POLICY_NAME:my_policy)(TOKEN_TYPE:THEME)(MAX_FEATURES:3000)"
```

Related Topics

- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.
- Specify Transformation Instructions for an Attribute
  You can pass transformation instructions for an attribute by defining a transformation list.
- *Oracle Database PL/SQL Packages and Types Reference*
- ALL_MINING_MODEL_ATTRIBUTES

39

Integration of ONNX Runtime

Learn about ONNX Runtime that enables you to use ONNX models for machine learning tasks.

39.1 About ONNX

ONNX is an open-source format designed for machine learning models. It ensures cross-platform compatibility. This format also supports major languages and frameworks, facilitating efficient model exchange.

The ONNX format allows for model serialization. It simplifies the exchange of models across various platforms. These platforms include cloud, web, edge, and mobile experiences on Microsoft Windows, Linux, Mac, iOS, and Android. ONNX models also offer flexibility to export and import model in many languages such as Python, C++, C#, and Java to name a few. The ONNX format is useful for compute-heavy tasks such as training machine learning models and data processing that often uses trained models. Many leading machine learning development frameworks such as TensorFlow, Pytorch, and Scikit-learn, offer the capability to convert models into the ONNX format.

Once you represent the models in the ONNX format, you can run them with the ONNX Runtime. The architecture of the ONNX Runtime is adaptable, enabling providers to modify or enhance how some operations are implemented to make better use of particular hardware, such as, Graphical Processing Units (GPUs), Single Instruction Multiple Data (SIMD) instruction sets or specialized libraries. To learn more on ONNX Runtime, see https://onnxruntime.ai/docs/.

The ONNX Runtime integration with Oracle Database lets you import ONNX-formatted models, including embedding models. To support embedding models, Oracle Machine Learning has introduced a new machine learning technique called *embedding*. If you do not have a pretrained model in ONNX format, Oracle offers a Python utility package that downloads a pretrained model, converts the model to ONNX format augmented with pre-processsing and post-processing operations and imports the ONNX format model to Oracle Database. To learn more on the Python utility tool, see Convert Pretrained Models to ONNX Format.

Oracle supports ONNX Runtime version 1.15.1.

39.1.1 Supported Machine Learning Functions for ONNX Runtime

Describes the supported machine learning functions to import pretrained models and perform scoring.

The following are the supported machine learning functions:

- Classification
- Clustering
- Embedding

- Regression

39.1.2 Supported Attribute Data Types

Discover the supported ONNX input data types mapped to SQL data types.

Data Type	SQL Type	Supported ONNX Data Type
Numerical	`BINARY_DOUBLE` `NUMBER`	`float`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`
Categorical	`VARCHAR`	For `VARCHAR` type: `string`
Text	`VARCHAR2` `CLOB`	`string`
Vectors	`VECTOR(float32,`*<dimension>*`)`	`float`

The following data types are not supported:

- `complex64`, `complex128`
- `float16`, `bfloat16`
- `fp8`
- `int4`, `uint4`

39.1.3 Supported Target Data Types

Discover the supported ONNX target data types mapped to SQL data types.

Depending on the machine learning function, different scoring functions are used. Different scoring function for same machine learning function can produce different data types. A few points to note:

- Classification models have different rules to determine the type of `PREDICTION` function to be used. If you are using `PREDICTION_PROBABILITY`, then `BINARY_DOUBLE` is returned. See labels in JSON Metadata Parameters for ONNX Models.
- For an embedding model, the `VECTOR_EMBEDDING` function returns a `VECTOR` type.
- For a regression model, `VARCHAR` is not a valid target type and `BINARY_DOUBLE` is returned.
- For a clustering model, if you are using `CLUSTERING_PROBABILITY` and `CLUSTER_DISTANCE`, then `BINARY_DOUBLE` is returned.

To learn more, see JSON Metadata Parameters for ONNX Models

Machine Learning Function	SQL Function	SQL Type	Supported ONNX Target Output
Regression	`PREDICTION`	`BINARY_DOUBLE`	`regressionOutput`
Classification	`PREDICTION`	`VARCHAR2`	`classificationLabelOutput`

Machine Learning Function	SQL Function	SQL Type	Supported ONNX Target Output
Classification	PREDICTION	NUMBER	classificationLabelOutput
Classification	PREDICTION_PROBABILITY	BINARY_DOUBLE	classificationProbOutput
Classification	PREDICTION_SET	set of (NUMBER , BINARY_DOUBLE) set of (target_type, BINARY_DOUBLE)	NA
Clustering	CLUSTER_PROBABILITY	BINARY_DOUBLE	clusteringProbOutput
Clustering	CLUSTER_DISTANCE	BINARY_DOUBLE	clusteringDistanceOutput
Clustering	CLUSTER_SET	set of (NUMBER , BINARY_DOUBLE)	NA
Embedding	VECTOR_EMBEDDING	VECTOR(float32, n)	embeddingOutput

39.1.4 Custom ONNX Runtime Operations

If you are looking to customize a pretrained embedding model by augmenting with pre-processing and post-processing operations, Oracle supports tokenization of an embedding model as a pre-processing operation and pooling and normalization as post-processing custom ONNX Runtime operations for version 1.15.1.

Oracle offers a Python utility that provides a mechanism to augment a pretrained model with tokenization, pooling and normalization. The Python utility can augment the model with pre-processing and post-processing operations and convert a pretrained model to an ONNX format. Models using any other custom operations will fail on import. For details on how to use the Python utility, see Convert Pretrained Models to ONNX Format.

39.1.5 Use PL/SQL Packages to Import Models

Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure or the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to import ONNX format models. You can then use the imported ONNX format models through a scoring function run by the in-database ONNX Runtime.

- To import a pretrained ONNX format model, use IMPORT_ONNX_MODEL Procedure or LOAD_ONNX_MODEL Procedure.
- To drop an ONNX model, use DROP_ONNX_MODEL. See also DROP_MODEL procedure.
- A complete step-by-step example that illustrates these procedures is in Import ONNX Models and Generate Embeddings.

The `DBMS_DATA_MINING.RENAME_MODEL` procedure is also supported.

Most of the existing Oracle Machine Learning for SQL APIs are available to the ONNX models. As partitioning is not applicable for external pretrained models, ONNX models do not support the following procedures:

- `ADD_PARTITION`
- `DROP_PARTITION`
- `ADD_COST_MATRIX`
- `REMOVE_COST_MATRIX`

Related Topics

- Summary of DBMS_DATA_MINING Subprograms

39.1.6 Supported SQL Scoring Functions

Supported scoring functions for in-database scoring of machine learning models imported in the ONNX format are listed.

Machine Learning Technique	Operator	Supported	Return Type
Embedding	`VECTOR_EMBEDDING`	always	`VECTOR(<dimensions , FLOAT32>)` The number of dimensions of the output vector of a `VECTOR_EMBEDDING` operator is defined by the embedding models.
Regression	`PREDICTION`	always	Data type of the target. For regression, the data type is converted to `BINARY_DOUBLE` SQL type.
Classification	`PREDICTION`	always	Data type of the target.
Classification	`PREDICTION_PROBABILITY`	always	`BINARY_DOUBLE`
Classification	`PREDICTION_SET`	always	Set of `( t, NUMBER , BINARY_DOUBLE )` where `t` is the data type of the target.
Clustering	`CLUSTER_ID`	only if `clusteringProbOutput` is specified	`NUMBER`
Clustering	`CLUSTER_PROBABILITY`	only if `clusteringProbOutput` is specified	`BINARY_DOUBLE`
Clustering	`CLUSTER_SET`	only if `clusteringProbOutput` is specified	Set of `( NUMBER, BINARY_DOUBLE )`

Machine Learning Technique	Operator	Supported	Return Type
Clustering	`CLUSTER_DISTANCE`	only if `clusteringDistanceOutput` is specified	`BINARY_DOUBLE`

> **Note:**
>
> You can define the outputs explicitly in the metadata or implicitly.
>
> - The metadata must explicitly specify how to find the result in the model output for some SQL scoring functions. For example, `CLUSTER_PROBABILITY` is supported only if `clusteringProbOutput` is specified in the metadata.
> - The system automatically assumes the output for a model with only one output if you don't specify it in the metadata.
> - If a scoring function does not comply according to the description provided, you will receive an ORA-40290 error when performing the scoring operation on your data. Additionally, any unsupported scoring functions will raise the ORA-40290 error.

To learn more about classification data types that are returned, see `labels` and `classificationLabelOutput` in JSON Metadata Parameters for ONNX Models.

Cost Matrix Clause

Specify a cost matrix directly within the `PREDICTION` and `PREDICTION_SET` scoring functions. To learn more about Cost Matrix, see *Oracle Machine Learning for SQL Concepts*.

39.2 Examples of Using ONNX Models

The following examples use Iris data set to showcase loading and inference from ONNX format machine learning models for machine learning techniques such as Classification, Regression, and Clustering.

Iris is a flower and this data set has information such as petal length, sepal length, petal width, and sepal width collected from three types of Iris flowers: Sentosa, Versicolour, and Virginica.

These examples assume that the data set is available to the user.

ONNX Classification Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX models.

Example: Specifying JSON Metadata for Classification Models

The following example illustrates JSON metadata parameters with Classification as the function. Assume the model has an output named `probabilities` for the probability of the prediction. To use the PREDICTION_PROBABILITY scoring function, you must set the field classificationProbOutput to the name of the model output that holds the probability.

```
BEGIN
LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities"}'));
END;
/
```

Example: Specifying labels in JSON Metadata for Classification Models

The following example illustrates how you can specify custom labels in the JSON metadata.

```
BEGIN
LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "labels": ["Setosa", "Versicolour", "Virginica"]}'));
END;
/
```

You can use the PREDICTION and PREDICTION_PROBABILITY functions for inference or scoring:

```
SELECT
    iris.*,
    PREDICTION(doc_model USING *) as predicted_species_id,
    PREDICTION_PROBABILITY(doc_model, 'setosa' USING *) as
setosa_probability
FROM iris;
```

The query predicts iris species and the probability of *setosa* species using the iris data set. The data from iris table is used in a SELECT query to predict a species ID and the probability that the species is *setosa* using a machine learning model named doc_model. The PREDICTION function predicts the species based on the attributes in the table, and the PREDICTION_PROBABILITY function computes the probability that the predicted species is *setosa*. The result includes all columns from the iris view along with the predicted species ID and the probability of the species being *setosa*.

Example: Specifying input in JSON Metadata for Classification Models

The following example illustrates how you can specify input attribute names that map to the actual ONNX model input names. This example assumes a model with four inputs named SEPAL_LENGTH, SEPAL_WIDTH, PETAL_LENGTH, and PETAL_WIDTH. You can specify alternative input attribute names using the JSON metadata as shown in this example. Here, each input is assumed to be a tensor with a dimension of 1. The input field must be a JSON object where each field is a model input name (For example, SEPAL_LENGTH), and its value is a JSON array sized according to the tensor's dimension (here, 1) with one attribute name per element in the array.

```
BEGIN LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "input": { "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"],
                  "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                  "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                  "PETAL_WIDTH": ["PETAL_WIDTH_CM"] } }'));
END;
/
```

You can also have a different order of the columns as input.

```
BEGIN LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
    JSON('{"function" : "classification",
           "classificationProbOutput": "probabilities",
           "input": { "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                   "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                   "PETAL_WIDTH": ["PETAL_WIDTH_CM"],
                "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"] } }'));
END;
/
```

Example: Specifying a Single input With Four Dimensions

Here is an example where the model has a single input tensor named x with four dimensions. The corresponding JSON metadata for this scenario is:

```
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "input": { "x": ["SEPAL_LENGTH_CM",
                        "SEPAL_WIDTH_CM",
                        "PETAL_LENGTH_CM",
                        "PETAL_WIDTH_CM"]
                     }'));
```

You can use PREDICTION and PREDICTION_PROBABILITY functions for inference or scoring.

```
WITH
dummy_iris AS (
    SELECT
    4.5 as petal_length_cm,
    1.5 as petal_width_cm,
    4.3 as sepal_length_cm,
    2.9 as sepal_width_cm
    FROM iris
)
SELECT
    dummy_iris.*,
    PREDICTION(doc_model USING *) as predicted_species_id,
```

```
    PREDICTION_PROBABILITY(doc_model 'setosa' USING *) as
setosa_probability
FROM dummy_iris;
```

The query predicts `iris` species and the probability of *setosa* species using specified attributes in a temporary data set. The query creates a temporary `dummy_iris` view with attributes values set. This temporary view is then used in a `SELECT` query to predict a species ID and the probability that the species is *setosa* using a machine learning model named `doc_model`. The `PREDICTION` function predicts the species based on the attributes provided, and the `PREDICTION_PROBABILITY` function computes the probability that the predicted species is *setosa*. The result includes all columns from the `dummy_iris` view along with the predicted species ID and the probability of the species being *setosa*.

Example: Specifying defaultOnNull in JSON Metadata for Classification Models

The following examples illustrates how you can specify `defaulOnNull` provides default values to be used for specific attributes when their values are NULL in the data set. Use the names `SEPAL_LENGTH`, `SEPAL_WIDTH`, `PETAL_LENGTH`, and `PETAL_WIDTH` as fields in the `defaultOnNull` object, which are the assumed input attribute names for a ONNX model with four inputs. These names serve as the default input attribute names, so you can use them as fields in the `defaultOnNull`.

```
BEGIN LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
    JSON('{"function" : "classification",
           "classificationProbOutput": "probabilities",
           "defaultOnNull": {"SEPAL_LENGTH": "5.1",
           "SEPAL_WIDTH": "3.5",
           "PETAL_LENGTH": "1.4",
           "PETAL_WIDTH": "0.2"}}'));
END;
/
```

- `"SEPAL_LENGTH": "5.1"`: If the sepal length is null, use 5.1 as the default value.
- `"SEPAL_WIDTH": "3.5"`: If the sepal width is null, use 3.5 as the default value.
- `"PETAL_LENGTH": "1.4"`: If the petal length is null, use 1.4 as the default value.
- `"PETAL_WIDTH": "0.2"`: If the petal width is null, use 0.2 as the default value.

Example: Specifying input and defaultOnNull JSON Metadata for Classification Models

Here is a combined example of specifying `input` and `defaultOnNull` values. This example uses the values that were illustrated in the earlier examples where `input` and `defaultOnNull` values are specified:

```
JSON('{"function" : "classification",
               "classificationProbOutput": "probabilities",
               "input": { "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                          "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                          "PETAL_WIDTH": ["PETAL_WIDTH_CM"],
                          "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"] },
```

```
                              "defaultOnNull": {"SEPAL_LENGTH_CM": "5.1",
                                                "SEPAL_WIDTH_CM": "3.5"}}')
```

ONNX Clustering Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX models.

Example: Specifying JSON Metadata for Clustering Models

The following example illustrates JSON metadata parameters with Clustering as the function. Assume the model has an output named `probabilities` for the probability of the prediction. To use the `CLUSTER_PROBABILITY` scoring function, you must set the field `clusteringProbOutput` to the name of the model output that holds the probability.

```
BEGIN
LOAD_ONNX_MODEL('clustering_model.onnx','doc_model',
    JSON('{"function": "clustering",
           "clusteringProbOutput": "probabilities"
        }
    ')
);
END;
/
```

You can use `CLUSTER_ID` and `CLUSTER_PROBABILITY` functions for inference or scoring.

```
SELECT
    iris.*,
    CLUSTER_ID(doc_model USING *) as cluster_id,
    CLUSTER_PROBABILITY(doc_model, 1 USING *) as cluster_1_probability
FROM iris;
```

This query predicts the cluster assignments and the probabilities of belonging to a specific cluster for each record of the `iris` data set. The query retrieves all columns of each record (`iris.*`) and applies the clustering model named `doc_model` to each record of the `iris` data set and predicts the cluster ID. The `USING *` clause tells the model to use all available columns in the `iris` table for this prediction. The `CLUSTER_PROBABILITY(doc_model, 1 USING *) as cluster_1_probability` part of the query calculates the probability that each record belongs to cluster 1, according to the `doc_model` from the `iris` data set. This provides insights into how likely each record is to be part of cluster 1, giving a quantitative measure of membership strength.

Example: Specifying clusteringDistanceOutput in JSON Metadata for Clustering Models

The following example illustrates how you can specify `clusteringDistanceOutput` and for ONNX Clustering models.

In this model, an output tensor named `distances` provides distances for the input, which is a single tensor named `float_input` with a dimension of 4. The JSON metadata `input` field must map attribute names to entries of the tensor, such as "SEPAL_LENGTH", "SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH".

```
BEGIN
LOAD_ONNX_MODEL('clustering_model.onnx', 'doc_model',
JSON('{"function" : "clustering",
"clusteringDistanceOutput": "distances",
"normalizeProb": "softmax",
"input": { "float_input": ["SEPAL_LENGTH", "SEPAL_WIDTH",
"PETAL_LENGTH", "PETAL_WIDTH"] }
      }')
);
END;
/
```

You can use CLUSTER_DISTANCE function for inference or scoring. These SQL queries utilize clustering models to predict cluster distances from the IRIS data set.

```
SELECT CLUSTER_DISTANCE(doc_model USING *) AS predicted_target_value,
CLUSTER_DISTANCE (doc_model,1 USING *) AS dist1,
CLUSTER_DISTANCE (doc_model,2 USING *) AS dist2,
CLUSTER_DISTANCE (doc_model,3 USING *) AS dist3
FROM IRIS
ORDER BY ID
FETCH NEXT 10 ROWS ONLY;
```

Here, the query focuses on understanding the physical distance of data points from cluster centroids, which is particularly useful for identifying outliers or for performing detailed cluster analysis. The query calculates the distance of each record in the IRIS data set from the centroids of different clusters using the doc_model. The USING * syntax indicates that the model must use all available columns of the IRIS data set for making the prediction. CLUSTER_DISTANCE(doc_model, n USING *) computes the distance from cluster n (n being 1, 2, and 3 in this query). Each distance is selected as a separate column (dist1, dist2, dist3).

The output is limited to the first 10 rows of the result set ordered by the ID column of the IRIS table.

Example: Specifying clusteringProbOutput and normalizeProb in JSON Metadata for Clustering Models

The following example illustrates how you can specify clusteringProbOutput and normalizeProb for ONNX Clustering models.

```
BEGIN
LOAD_ONNX_MODEL('clustering_model.onnx', 'doc_model',
               JSON('{"function" :
               "clustering",
               "clusteringProbOutput": "probabilities",
                        "normalizeProb" : "softmax",
               "input": { "float_input": ["SEPAL_LENGTH",
"SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH"] } }')
      );
```

```
END;
/
```

You can use CLUSTER_PROBABILITY and CLUSTER_SET functions for inference or scoring:

```
SELECT CLUSTER_ID (doc_model USING *) AS predicted_target_value,
                    CLUSTER_PROBABILITY (doc_model,1 USING *) AS prob1,
                    CLUSTER_PROBABILITY (doc_model,2 USING *) AS prob2,
                    CLUSTER_PROBABILITY (doc_model,3 USING *) AS prob3
                    FROM IRIS
                    ORDER BY ID
                    FETCH NEXT 10 ROWS ONLY;
```

In this case, a clustering model is used to predict the cluster IDs and associated probabilities for records from the IRIS data set. Because the JSON metadata specifies softmax for the normalizeProb field, the model applies softmax normalization to the probabilities before returning them as the result of the CLUSTER_PROBABILITY scoring operator.

The SQL query selects CLUSTER_ID column from the IRIS table and adds a new column, predicted_target_value, which contains predictions made by the doc_model. The USING * syntax means that all columns of the current row are used as input features for the doc_model model to predict the value as predicted_target_value. The result of this prediction is then included as a new column in the output of the query.

CLUSTER_PROBABILITY(model, n USING *): Computes the probability that the record belongs to cluster n (n being 1, 2, and 3 in this query). This is done for three different clusters, and each probability is selected as a separate column (prob1, prob2, prob3).

The output is limited to the first 10 rows of the result set ordered by the ID column of the IRIS table.

```
SELECT S.CLUSTER_ID, S.PROBABILITY
 FROM (SELECT CLUSTER_SET(doc_model USING *) pset
          FROM IRIS ORDER BY ID) T,
       TABLE(T.pset) S
FETCH NEXT 10 ROWS ONLY;
```

The CLUSTER_SET query generates a set of cluster data using the doc_model. The resultant column pset represents all possible cluster assignments for each record, which includes cluster IDs and their respective probabilities ordered by the ID column. The SELECT S.CLUSTER_ID, S.PROBABILITY part of the query selects the cluster ID and probability from the resultant column set. The output is limited to the first 10 rows of the result set.

ONNX Regression Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX Regression models. All examples assume an ONNX model that has one output named regressionOutput and four input tensors of dimension 1 whose name match exactly the name of the IRIS table columns, namely, SEPAL_LENGTH, SEPAL_WIDTH, PETAL_LENGTH, PETAL_WIDTH.

Example: Specifying JSON Metadata for Regression Models

The following is a simple example illustrating JSON metadata parameters with Regression as the function. Assume the ONNX model features one output named `regressionOutput` and four input tensors of dimension 1, whose names match exactly after the `IRIS` table columns (`"SEPAL_LENGTH", "SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH"`). The JSON metadata can be as simple as the following:

```
BEGIN LOAD_ONNX_MODEL(
    'regression_model.onnx',
    'doc_model',
    JSON('{"function": "regression"}
    ')
);
END;
/
```

You can use the `PREDICTION` function for inference or scoring:

```
SELECT
    iris.*,
    PREDICTION(doc_model USING *) as predicted_petal_width_cm
FROM iris;
```

In this case, the SQL query selects all columns from the `iris` table and adds a new column, `predicted_petal_width_cm`, which contains predictions made by the `doc_model`. The `USING *` syntax means that all columns of the current row are used as input features for the `doc_model` model to predict the value of `PETAL_WIDTH` as `predicted_petal_width_cm`. The result of this prediction is then included as a new column in the output of the query.

Example: Specifying input and defaultOnNull in JSON Metadata for Regression Models

The following example illustrates how you can specify input attribute names that map to the actual ONNX model input names. The `defaulOnNull` providing default values to be used for specific attributes when their values are NULL in the data set.

```
BEGIN LOAD_ONNX_MODEL('regression_model.onnx','doc_model',
    JSON('{"function": "regression",
            "input": {
                "SEPAL_LENGTH": ["dummy_sepal_length_cm"],
                "SEPAL_WIDTH": ["dummy_sepal_width_cm"]
                      },
            "defaultOnNull": {
                "dummy_sepal_length_cm": "5.1",
                "dummy_sepal_width_cm": "3.5",
                 }
           }
        ')
);
```

```
END;
/
```

You can use the PREDICTION function for inference or scoring.

```
WITH
dummy_iris AS (
    SELECT
    (CASE WHEN petal_length > 5 THEN 4.9 ELSE NULL END)
        as dummy_sepal_length_cm,
    (CASE WHEN petal_length < 4 THEN 2.5 ELSE NULL END)
        as dummy_sepal_width_cm,
    petal_length
    petal_width
    FROM iris
)
SELECT
    dummy_iris.*,
    PREDICTION(doc_model USING *) as predicted_petal_width_cm
FROM dummy_iris;
```

In this case, a temporary `dummy_iris` table is created with three columns: `dummy_sepal_length_cm`, `dummy_sepal_width_cm`, and `petal_length`. The values of the `dummy_sepal_length_cm` and `dummy_sepal_width_cm` are based on `petal_length` values of the `iris` table. If `petal_length` is greater than 5, `dummy_sepal_length_cm` is set to 4.9, otherwise it is NULL. If `petal_length` is less than 4, `dummy_sepal_width_cm` is set to 2.5, otherwise it remains NULL.

Then the `SELECT` query retrieves all columns from the `dummy_iris` table and uses the `doc_model` to predict `petal_width`, adding this prediction as a new column named `predicted_petal_width_cm`. The model uses the derived dummy columns, `petal_length` and `petal_width` for its predictions.

See Also:

- LOAD_ONNX_MODEL in *Oracle Database PL/SQL Packages and Types Reference*
- Supported SQL Scoring Functions

40 Administrative Tasks for Oracle Machine Learning for SQL

Explains how to perform administrative tasks related to Oracle Machine Learning for SQL.

- Install and Configure a Database for Oracle Machine Learning for SQL
- Upgrade or Downgrade Oracle Machine Learning for SQL
- Export and Import Oracle Machine Learning for SQL Models
- Secure
- Audit and Add Comments to Oracle Machine Learning for SQL Models

40.1 Install and Configure a Database for Oracle Machine Learning for SQL

You can install and configure a database for Oracle Machine Learning for SQL by following the listed steps.

- About Installation
- Database Tuning Considerations for Oracle Machine Learning for SQL

40.1.1 About Installation

Oracle Machine Learning components associated with Oracle Database are included with the database license.

To install Oracle Database, follow the installation instructions for your platform. Choose a Data Warehousing configuration during the installation.

Oracle Data Miner, the graphical user interface to Oracle Machine Learning for SQL, is an extension to Oracle SQL Developer. Instructions for downloading SQL Developer and installing the Data Miner repository are available on https://www.oracle.com/database/technologies/odmrinstallation.html.

To perform machine learning activities, you must be able to log on to the Oracle Database, and your user ID must have the database privileges described in Grant Privileges for Oracle Machine Learning for SQL.

Related Topics

- `Oracle Data Miner`

See Also:

Install and Upgrade page of the Oracle Database online documentation library for your platform-specific installation instructions: `Oracle Database 23c Release`

40.1.2 Database Tuning Considerations for Oracle Machine Learning for SQL

Standard administrative practices can be followed to manage workload on the system when machine learning activities are running.

DBAs managing production databases that support Oracle Machine Learning for SQL must follow standard administrative practices as described in *Oracle Database Administrator's Guide*.

Building machine learning models and batch scoring of machine learning models tend to put a DSS-like workload on the system. Single-row scoring tends to put an OLTP-like workload on the system.

Database memory management can have a major impact on machine learning. The correct sizing of Program Global Area (PGA) memory is very important for model building, complex queries, and batch scoring. From a machine learning perspective, the System Global Area (SGA) is generally less of a concern. However, the SGA must be sized to accommodate real-time scoring, which loads models into the shared cursor in the SGA. In most cases, you can configure the database to manage memory automatically. To do so, specify the total maximum memory size in the tuning parameter `MEMORY_TARGET`. With automatic memory management, Oracle Database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands.

Most machine learning algorithms can take advantage of parallel execution when it is enabled in the database. Parameters in `INIT.ORA` control the behavior of parallel execution.

40.2 Upgrade or Downgrade Oracle Machine Learning for SQL

Upgrade and downgrade Oracle Machine Learning for SQL by following the steps listed.

40.2.1 Pre-Upgrade Steps

Pre-upgrade considerations.

Before upgrading, you must drop any machine learning models and machine learning activities that were created inOracle Data Miner.

40.2.2 Upgrade Oracle Machine Learning for SQL

You can upgrade your database by using the Database Upgrade Assistant (DBUA) or you can perform a manual upgrade using export/import utilities.

All models and machine learning metadata are fully integrated with the Oracle Database upgrade process whether you are upgrading from 19*c* or from earlier releases.

Upgraded models continue to work as they did in prior releases. Both upgraded models and new models that you create in the upgraded environment can make use of the new machine learning functionality introduced in the new release.

Related Topics

- Pre-Upgrade Steps
  Pre-upgrade considerations.
- *Oracle Database Upgrade Guide*

40.2.2.1 Use Database Upgrade Assistant to Upgrade Oracle Machine Learning for SQL

Oracle Database Upgrade Assistant provides a graphical user interface that guides you interactively through the upgrade process.

On Windows platforms, follow these steps to start the Upgrade Assistant:

1. Go to the Windows **Start** menu and choose the Oracle home directory.
2. Choose the **Configuration and Migration Tools** menu.
3. Launch the **Upgrade Assistant**.

On Linux platforms, run the `DBUA` utility to upgrade Oracle Database.

Related Topics

- *Oracle Database Upgrade Guide*

40.2.2.2 Use Export/Import to Upgrade Machine Learning Models

Use Export and Import functions of the Oracle Database to export the previously created models and import the models in an instance of Oracle Database version.

If required, you can use a less automated approach to upgrading machine learning models. You can export the models created in a previous version of Oracle Database and import them into an instance of the Oracle Database version.

40.2.2.2.1 Export/Import Oracle Machine Learning for SQL Models

Use the export and import functions of the Oracle Database to export the previously created models and import the models in an instance of Oracle Database version.

If required, you can use a less automated approach to upgrading machine learning models. You can export the models created in a previous version of Oracle Database and import them into an instance of the Oracle Database version.

To export models from an instance of a previous release of Oracle Database to a dump file, follow the instructions in Export and Import Oracle Machine Learning for SQL Models.

40.2.3 Post Upgrade Steps

Perform steps to view the upgraded database.

After upgrading the database, check the `DBA_MINING_MODELS` view in the upgraded database. The newly upgraded machine learning models must be listed in this view.

After you have verified the upgrade and confirmed that there is no need to downgrade, you must set the initialization parameter `COMPATIBLE` to `23.0.0`. In Oracle Database 23*c*, when the `COMPATIBLE` initialization parameter is not set in your parameter file, the `COMPATIBLE` parameter value defaults to `23.0.0`.

> **Note:**
>
> The `CREATE MINING MODEL` privilege must be granted to Oracle Machine Learning for SQL user accounts that are used to create machine learning models.

Related Topics

- Create an Oracle Machine Learning for SQL User
  An OML4SQL user is a database user account that has privileges for performing machine learning activities.
- Secure
  You can create an Oracle Machine Learning for SQL user and grant necessary privileges by following the steps listed.

40.2.4 Downgrade Oracle Machine Learning for SQL

Before downgrading the Oracle database back to the previous version, ensure that no models are present.

Use the `DBMS_DATA_MINING.DROP_MODEL` routine to drop the models before downgrading. If you do not do this, the database downgrade process terminates.

Issue the following SQL statement in `SYS` to verify the downgrade:

```
SQL>SELECT o.name FROM sys.model$ m, sys.obj$ o
                 WHERE m.obj#=o.obj# AND m.version=2;
```

40.3 Export and Import Oracle Machine Learning for SQL Models

You can export machine learning models to move models to a different Oracle Database instance, such as from a development database to a production database.

The `DBMS_DATA_MINING` package includes procedures for migrating machine learning models between database instances.

`EXPORT_MODEL` exports a single model or list of models to a dump file so it can be imported, queried, and scored in a separate Oracle Machine Learning database instance.

`IMPORT_MODEL` takes the dump file and creates the model in the destination database.

`EXPORT_SERMODEL` exports a single model to a serialized `BLOB` so it can be imported and scored in a separate Oracle Machine Learning database instance or to OML Services.

`IMPORT_SERMODEL` takes the serialized `BLOB` and creates the model in the destination database.

Related Topics

- `EXPORT_MODEL`
- `IMPORT_MODEL`
- `EXPORT_SERMODEL`
- `IMPORT_SERMODEL`

40.3.1 About Exporting Models

As a result of building models, each model has a set of model detail views that provide information about the model, such as model statistics for evaluation. The user can query these model detail views. With serialized models, only the model data and metadata required for scoring are available in the serialized model. This is more compact and transfers faster to the destination environment than dump files produced by the `EXPORT_MODEL` procedure.

To retain complete model details, use the `DMBS_DATA_MINING.EXPORT_MODEL` procedure and the `DBMS_DATA_MINING.IMPORT_MODEL` procedure. Serialized model export only works with models that produce scores. Specifically, it doesn't support Attribute Importance, Association Rules, Exponential Smoothing, or O-Cluster (although O-Cluster does allow scoring). Use `EXPORT_MODEL` to export these models and scenarios when full model details are needed.

Related Topics

- EXPORT_MODEL Procedure
- IMPORT_MODEL Procedure

40.3.2 About Oracle Data Pump

Use the command-line clients of Oracle Data Pump to export and import schemas or databases.

Oracle Data Pump consists of two command-line clients and two PL/SQL packages. The command-line clients, `expdp` and `impdp`, provide an easy-to-use interface to the Data Pump export and import utilities. You can use `expdp` and `impdp` to export and import entire schemas or databases respectively.

The Data Pump export utility writes the schema objects, including the tables and metadata that constitute machine learning models, to a dump file set. The Data Pump import utility retrieves the schema objects, including the model tables and metadata, from the dump file set and restores them in the target database.

`expdp` and `impdp` cannot be used to export/import individual machine learning models.

See Also:

Oracle Database Utilities for information about Oracle Data Pump and the `expdp` and `impdp` utilities

40.3.3 Options for Exporting and Importing Oracle Machine Learning for SQL Models

Lists options for exporting and importing machine learning models.

Options for exporting and importing machine learning models are described in the following table.

Table 40-1 Export and Import Options for Oracle Machine Learning for SQL

Task	Description
Export or import a full database	(DBA only) Use `expdp` to export a full database and `impdp` to import a full database. All machine learning models in the database are included.
Export or import a schema	Use `expdp` to export a schema and `impdp` to import a schema. All machine learning models in the schema are included.
Export or import models within a database or between databases	Use `DBMS_DATA_MINING.EXPORT_MODEL` to export one or more models and `DBMS_DATA_MINING.IMPORT_MODEL` to import one or more models. These procedures can export and import a single machine learning model, all machine learning models, or machine learning models that match specific criteria. To import models, you must have the `CREATE TABLE`, `CREATE VIEW`, and `CREATE MINING MODEL` privileges.
Export or import individual models to or from a remote database	Use a database link to export individual models to a remote database or import individual models from a remote database. A database link is a schema object in one database that enables access to objects in a different database. The link must be created before you run `EXPORT_MODEL` or `IMPORT_MODEL`. To create a private database link, you must have the `CREATE DATABASE LINK` system privilege. To create a public database link, you must have the `CREATE PUBLIC DATABASE LINK` system privilege. Also, you must have the `CREATE SESSION` system privilege on the remote Oracle Database. Oracle Net must be installed on both the local and remote Oracle Databases.

Table 40-1 (Cont.) Export and Import Options for Oracle Machine Learning for SQL

Task	Description
Serialized model export and import	Starting from Oracle Database 18c, the serialized model format was introduced as a lightweight approach to support scoring. The `DBMS_DATA_MINING.EXPORT_SERMODEL` procedure exports a single model to a serialized `BLOB` so it can be imported and scored in a separate Oracle Machine Learning (OML) database instance or to OML Services. `DBMS_DATA_MINING.IMPORT_SERMODEL` takes the serialized `BLOB` and creates the model in the target database.

Related Topics

- IMPORT_MODEL Procedure
- EXPORT_MODEL Procedure
- *Oracle Database SQL Language Reference*

40.3.4 Directory Objects for EXPORT_MODEL and IMPORT_MODEL

Learn how to use directory objects to identify the location of the dump file set containing the models.

`EXPORT_MODEL` and `IMPORT_MODEL` use a directory object to identify the location of the dump file set. A directory object is a logical name in the database for a physical directory on the host computer.

To export machine learning models, you must have write access to the directory object and to the file system directory that it represents. To import machine learning models, you must have read access to the directory object and to the file system directory. Also, the database itself must have access to file system directory. You must have the `CREATE ANY DIRECTORY` privilege to create directory objects.

The following SQL command creates a directory object named `omldir`. The file system directory that it represents must already exist and have shared read/write access rights granted by the operating system. For example, if the directory path is `/home/omluser`, the command is:

```
CREATE OR REPLACE DIRECTORY omldir AS '/home/omluser';
```

The following SQL command gives user `omluser` both read and write access to `omldir`.

```
GRANT READ,WRITE ON DIRECTORY omldir TO OMLUSER;
```

Related Topics

- *Oracle Database SQL Language Reference*

40.3.5 Use EXPORT_MODEL and IMPORT_MODEL

The examples illustrate various export and import scenarios with `EXPORT_MODEL` and `IMPORT_MODEL`.

The examples use the directory object `OMLDIR` shown in Example 40-1 and two schemas, `DM1` and `DM2`. Both schemas have machine learning privileges. `DM1` has two models. `DM2` has one model.

The DM1 schema has the following models:

- The EM_SH_CLUS_SAMPLE model: it is created by the oml4sql-clustering-expectation-maximization.sql example.
- The DT_SH_CLAS_SAMPLE model: it is created by the oml4sql-classification-decision-tree.sql example.

The DM2 schema has the SVD_SH_SAMPLE model and is created by the oml4sql-singular-value-decomposition.sql. In the following code, models in DM1 schema are displayed.

```
SELECT owner, model_name, mining_function, algorithm FROM all_mining_models
where OWNER='DM1';
```

The output is as follows:

```
OWNER      MODEL_NAME           MINING_FUNCTION      ALGORITHM
---------- -------------------- --------------------
--------------------------
DM1        EM_SH_CLUS_SAMPLE    CLUSTERING
EXPECTATION_MAXIMIZATION
DM1        DT_SH_CLAS_SAMPLE    CLASSIFICATION       DECISION_TREE
```

Example 40-1 Creating the Directory Object

```
-- connect as system user
CREATE OR REPLACE DIRECTORY OMLDIR AS '/home/oracle';
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM1;
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM2;
SELECT * FROM all_directories WHERE directory_name = 'OMLDIR';



OWNER      DIRECTORY_NAME             DIRECTORY_PATH
---------- --------------------------
----------------------------------------
SYS        OMLDIR                      /home/omluser
```

Example 40-2 Exporting All Models From DM1

```
-- connect as DM1
BEGIN
  dbms_data_mining.export_model (
                 filename =>   'all_DM1',
                 directory =>  'OMLDIR');
END;
/
```

A log file and a dump file are created in /home/omluser, the physical directory associated with OMLDIR. The name of the log file is dm1_exp_11.log. The name of the dump file is all_dm101.dmp.

Example 40-3 Importing the Models Back Into DM1

The models that were exported in Example 40-2 still exist in `DM1`. Since an import does not overwrite models with the same name, you must drop the models before importing them back into the same schema.

```
BEGIN
  dbms_data_mining.drop_model('EM_SH_CLUS_SAMPLE');
  dbms_data_mining.drop_model('DT_SH_CLAS_SAMPLE');
  dbms_data_mining.import_model(
                  filename => 'all_dm101.dmp',
                  directory => 'OMLDIR');
END;
/
SELECT model_name FROM user_mining_models;


MODEL_NAME
------------------------------
DT_SH_CLAS_SAMPLE
EM_SH_CLUS_SAMPLE
```

Example 40-4 Importing Models Into a Different Schema

In this example, the models that were exported from `DM1` in Example 40-2 are imported into `DM2`. The `DM1` schema uses the `USER1` tablespace; the `DM2` schema uses the `USER2` tablespace.

```
-- CONNECT as sysdba
BEGIN
  dbms_data_mining.import_model (
                  filename => 'all_d101.dmp',
                  directory => 'OMLDIR',
                  schema_remap => 'DM1:DM2',
                  tablespace_remap => 'USER1:USER2');
END;
/
-- CONNECT as DM2
SELECT model_name from user_mining_models;


MODEL_NAME
--------------------------------------------------------------------------------
---
SVD_SH_SAMPLE
EM_SH_CLUS_SAMPLE
DT_SH_CLAS_SAMPLE
```

Example 40-5 Exporting Specific Models

You can export a single model, a list of models, or a group of models that share certain characteristics.

```
-- Export the model named dt_sh_clas_sample
EXECUTE dbms_data_mining.export_model (
             filename => 'one_model',
             directory =>'OMLDIR',
             model_filter => 'name in (''DT_SH_CLAS_SAMPLE'')');
-- one_model01.dmp and dm1_exp_37.log are created in /home/omluser
```

```
-- Export Decision Tree models
EXECUTE dbms_data_mining.export_model(
             filename => 'algo_models',
             directory => 'OMLDIR',
             model_filter => 'ALGORITHM_NAME IN (''DECISION_TREE'')');
-- algo_model01.dmp and dm1_exp_410.log are created in /home/omluser

-- Export clustering models
EXECUTE dbms_data_mining.export_model(
             filename =>'func_models',
             directory => 'OMLDIR',
             model_filter => 'FUNCTION_NAME = ''CLUSTERING''');
-- func_model01.dmp and dm1_exp_513.log are created in /home/omluser
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

40.3.6 EXPORT and IMPORT Serialized Models

From Oracle Database Release 18c onwards, `EXPORT_SERMODEL` and `IMPORT_SERMODEL` procedures are available to export or import serialized models to or from a database.

The serialized format allows the models to be moved to another database instance or OML Services for scoring. The model is exported to a serialized `BLOB` . The import routine takes the serialized content in the `BLOB` and the name of the model to be created with the content.

Related Topics

- EXPORT_SERMODEL Procedure
- IMPORT_SERMODEL Procedure

40.3.7 Import From PMML

You can import regression models represented in Predictive Model Markup Language (PMML).

PMML is an XML-based standard specified by the Data Mining Group (`https://www.dmg.org`). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Machine Learning for SQL supports the core features of PMML 3.1 for regression models.

You can import regression models represented in PMML. The models must be of type `RegressionModel`, either linear regression or binary logistic regression.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

40.4 Secure

You can create an Oracle Machine Learning for SQL user and grant necessary privileges by following the steps listed.

- Create an Oracle Machine Learning for SQL User

- System Privileges for Oracle Machine Learning for SQL
- Object Privileges for Oracle Machine Learning for SQL Models

40.4.1 Create an Oracle Machine Learning for SQL User

An OML4SQL user is a database user account that has privileges for performing machine learning activities.

Example 40-6 shows how to create a database user. Example 40-7 shows how to assign machine learning privileges to the user.

> **Note:**
>
> To create a user for the OML4SQL examples, you must run two configuration scripts as described in Install the OML4SQL Examples.

Example 40-6 Creating a Database User in SQL*Plus

1. Log in to SQL*Plus with system privileges.

   ```
   Enter user-name: sys as sysdba
   Enter password: password
   ```

2. To create a user named `oml_user`, type these commands. Specify a password of your choosing.

   ```
   CREATE USER oml_user IDENTIFIED BY password
          DEFAULT TABLESPACE USERS
          TEMPORARY TABLESPACE TEMP
          QUOTA UNLIMITED ON USERS;
   Commit;
   ```

   The `USERS` and `TEMP` tablespaces are included in Oracle Database. `USERS` is used mostly by demo users; it is appropriate for running the examples described in About the OML4SQL Examples. `TEMP` is the temporary tablespace that is shared by most database users.

   > **Note:**
   >
   > Tablespaces for OML4SQL users must be assigned according to standard DBA practices, depending on system load and system resources.

3. To log in as `oml_user`, enter the following.

   ```
   CONNECT oml_user
   Enter password: password
   ```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the complete syntax of the `CREATE USER` statement

40.4.1.1 Grant Privileges for Oracle Machine Learning for SQL

The `CREATE MINING MODEL` is a privilege that you must have to create and perform operations on your model. Some other machine learning privileges can be assigned by issuing `GRANT` statements.

You must have the `CREATE MINING MODEL` privilege to create models in your own schema. You can perform any operation on models that you own. This includes applying the model, adding a cost matrix, renaming the model, and dropping the model.

The `GRANT` statements in the following example assign a set of basic machine learning privileges to the `oml_user` account. Some of these privileges are not required for all machine learning activities, however it is prudent to grant them all as a group.

Additional system and object privileges are required for enabling or restricting specific machine learning activities.

The following table lists the system privileges required for running the OML4SQL examples.

Table 40-2 System Privileges Granted by dmshgrants.sql to the OML4SQL User

Privilege	Allows the OML4SQL User To
`CREATE SESSION`	Log in to a database session
`CREATE TABLE`	Create tables, such as the settings tables for `CREATE_MODEL`
`CREATE VIEW`	Create views, such as the views of tables in the `SH` schema
`CREATE MINING MODEL`	Create OML4SQL models
`EXECUTE ON ctxsys.ctx_ddl`	Run procedures in the `ctxsys.ctx_ddl` PL/SQL package; required for text mining

Example 40-7 Privileges Required for Machine Learning

This example grants the required privileges to the user oml_user.

```
GRANT CREATE SESSION TO oml_user;
GRANT CREATE TABLE TO oml_user;
GRANT CREATE VIEW TO oml_user;
GRANT CREATE MINING MODEL TO oml_user;
GRANT EXECUTE ON CTXSYS.CTX_DDL TO oml_user;
```

`READ` or `SELECT` privileges are required for data that is not in your schema. For example, the following statement grants `SELECT` access to the `sh.customers` table.

```
GRANT SELECT ON sh.customers TO oml_user;
```

40.4.2 System Privileges for Oracle Machine Learning for SQL

A system privilege confers the right to perform a particular action in the database or to perform an action on a type of schema objects. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

You can perform specific operations on machine learning models in other schemas if you have the appropriate system privileges. For example, `CREATE ANY MINING MODEL` enables you to create models in other schemas. `SELECT ANY MINING MODEL` enables you to apply models that reside in other schemas. You can add comments to models if you have the `COMMENT ANY MINING MODEL` privilege.

To grant a system privilege, you must either have been granted the system privilege with the `ADMIN OPTION` or have been granted the `GRANT ANY PRIVILEGE` system privilege.

The system privileges listed in the following table control operations on machine learning models.

Table 40-3 System Privileges for Oracle Machine Learning for SQL

System Privilege	Allows you to....
`CREATE MINING MODEL`	Create machine learning models in your own schema.
`CREATE ANY MINING MODEL`	Create machine learning models in any schema.
`ALTER ANY MINING MODEL`	Change the name or cost matrix of any machine learning model in any schema.
`DROP ANY MINING MODEL`	Drop any machine learning model in any schema.
`SELECT ANY MINING MODEL`	Apply a machine learning model in any schema, also view model details in any schema.
`COMMENT ANY MINING MODEL`	Add a comment to any machine learning model in any schema.
`AUDIT_ADMIN` role	Generate an audit trail for any machine learning model in any schema. (See *Oracle Database Security Guide* for details.)

Example 40-8 Grant System Privileges for Oracle Machine Learning for SQL

The following statements allow `oml_user` to score data and view model details in any schema as long as `SELECT` access has been granted to the data. However, `oml_user` can only create models in the `oml_user` schema.

```
GRANT CREATE MINING MODEL TO oml_user;
GRANT SELECT ANY MINING MODEL TO oml_user;
```

The following statement revokes the privilege of scoring or viewing model details in other schemas. When this statement is run, `oml_user` can only perform machine learning activities in the `oml_user` schema.

```
REVOKE SELECT ANY MINING MODEL FROM oml_user;
```

Related Topics

- Add a Comment to an Oracle Machine Learning for SQL Model
  You can add a comment to an OML4SQL model object using SQL `COMMENT` statement.
- *Oracle Database Security Guide*

40.4.3 Object Privileges for Oracle Machine Learning for SQL Models

Learn about machine learning object privileges.

An object privilege confers the right to perform a particular action on a specific schema object. For example, the privilege to delete rows from the `SH.PRODUCTS` table is an example of an object privilege.

You automatically have all object privileges for schema objects in your own schema. You can grant object privilege on objects in your own schema to other users or roles.

The object privileges listed in the following table control operations on specific machine learning models.

Table 40-4 Object Privileges for Oracle Machine Learning for SQL Models

Object Privilege	Allows you to....
`ALTER MINING MODEL`	Change the name or cost matrix of the specified machine learning model object.
`SELECT MINING MODEL`	Apply the specified machine learning model object and view its model details.

Example 40-9 Grant Object Privileges on Oracle Machine Learning for SQL Models

The following statements allow `oml_user` to apply the model `testmodel` to the `sales` table, specifying different cost matrixes with each apply. The user `oml_user` can also rename the model `testmodel`. The `testmodel` model and `sales` table are in the `sh` schema, not in the `oml_user` schema.

```
GRANT SELECT ON MINING MODEL sh.testmodel TO oml_user;
GRANT ALTER ON MINING MODEL sh.testmodel TO oml_user;
GRANT SELECT ON sh.sales TO oml_user;
```

The following statement prevents `oml_user` from renaming or changing the cost matrix of `testmodel`. However, `oml_user` can still apply `testmodel` to the `sales` table.

```
REVOKE ALTER ON MINING MODEL sh.testmodel FROM oml_user;
```

40.5 Audit and Add Comments to Oracle Machine Learning for SQL Models

Perform audit of Oracle Machine Learning for SQL model objects through SQL statements.

OML4SQL model objects support SQL `COMMENT` and `AUDIT` statements.

40.5.1 Add a Comment to an Oracle Machine Learning for SQL Model

You can add a comment to an OML4SQL model object using SQL `COMMENT` statement.

Comments can be used to associate descriptive information with a database object. You can associate a comment with a machine learning model using a SQL `COMMENT` statement.

```
COMMENT ON MINING MODEL schema_name.model_name IS string;
```

> **Note:**
>
> To add a comment to a model in another schema, you must have the `COMMENT ANY MINING MODEL` system privilege.

To drop a comment, set it to the empty `''` string.

The following statement adds a comment to the model `DT_SH_CLAS_SAMPLE` in your own schema.

```
COMMENT ON MINING MODEL dt_sh_clas_sample IS
          'Decision Tree model predicts promotion response';
```

You can view the comment by querying the catalog view `USER_MINING_MODELS`.

```
SELECT model_name, mining_function, algorithm, comments FROM user_mining_models;
```

The output is as follows:

```

MODEL_NAME        MINING_FUNCTION  ALGORITHM      COMMENTS
----------------- ---------------- --------------
-----------------------------------------------
DT_SH_CLAS_SAMPLE CLASSIFICATION   DECISION_TREE  Decision Tree model
predicts promotion response
```

To drop this comment from the database, issue the following statement:

```
COMMENT ON MINING MODEL dt_sh_clas_sample '';
```

> **See Also:**
>
> - Table 40-3
> - *Oracle Database SQL Language Reference* for details about SQL `COMMENT` statements

40.5.2 Audit Oracle Machine Learning for SQL Models

Use Oracle Database auditing system to audit models to track operations on machine learning models.

The Oracle Database auditing system is a powerful, highly configurable tool for tracking operations on schema objects in a production environment. The auditing system can be used to track operations on machine learning models.

> **Note:**
>
> To audit machine learning models, you must have the `AUDIT_ADMIN` role.

Unified auditing is documented in *Oracle Database Security Guide*. However, the full unified auditing system is not enabled by default. Instructions for migrating to unified auditing are provided in *Oracle Database Upgrade Guide*.

> **See Also:**
>
> - "Auditing Oracle Machine Learning for SQL Events" in *Oracle Database Security Guide* for details about auditing machine learning models
> - "Monitoring Database Activity with Auditing" in *Oracle Database Security Guide* for a comprehensive discussion of unified auditing in Oracle Database
> - "About the Unified Auditing Migration Process for Oracle Database" in *Oracle Database Upgrade Guide* for information about migrating to unified auditing
> - *Oracle Database Upgrade Guide*

41

Examples

The OML4SQL examples are available on GitHub and some scenarios with those examples are illustrated.

- About the OML4SQL Examples
- PL/SQL API
- Example: Predicting Likely Candidates for a Sales Promotion
- Example: Analyzing Preferred Customers
- Example: Segmenting Customer Data
- Example : Comparison of Texts Using an ESA Model

41.1 About the OML4SQL Examples

The OML4SQL examples illustrate typical approaches to data preparation, algorithm selection, algorithm tuning, testing, and scoring.

You can learn a great deal about the OML4SQL application programming interface from the OML4SQL examples. The examples are simple. They include extensive inline comments to help you understand the code. They delete all temporary objects on exit so that you can run the examples repeatedly without setup or cleanup.

The OML4SQL examples are available on GitHub at https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql/. Select the Database release (for example 23c) to see the examples.

The OML4SQL examples create a set of machine learning models in the user's schema. The following table lists the file name of the example and the `mining_function` value and algorithm the example uses.

Table 41-1 Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
`oml4sql-anomaly-detection-1class-svm.sql`	`CLASSIFICATION`	`ALGO_SUPPORT_VECTOR_MACHINE`
`oml4sql-anomaly-detection-em.sql`	`CLASSIFICATION`	`ALGO_EXPECTATION_MAXIMIZATION`
`oml4sql-association-rules.sql`	`ASSOCIATION`	`ALGO_APRIORI_ASSOCIATION_RULES`
`oml4sql-feature-extraction-cur.sql`	`ATTRIBUTE_IMPORTANCE`	`ALGO_CUR_DECOMPOSITION`
`oml4sql-classification-decision-tree.sql`	`CLASSIFICATION`	`ALGO_DECISION_TREE`
`oml4sql-cross-validation-decision-tree.sql`	`CLASSIFICATION`	`ALGO_DECISION_TREE`

Table 41-1 (Cont.) Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
oml4sql-classification-glm.sql	CLASSIFICATION	ALGO_GENERALIZED_LINEAR_MODEL
oml4sql-time-series-mset.sql	CLASSIFICATION	ALGO_MSET_SPRT
oml4sql-classification-naive-bayes.sql	CLASSIFICATION	ALGO_NAIVE_BAYES
oml4sql-classification-neural-networks.sql	CLASSIFICATION	ALGO_NEURAL_NETWORK
oml4sql-classification-random-forest.sql	CLASSIFICATION	ALGO_RANDOM_FOREST
oml4sql-classification-regression-xgboost.sql	CLASSIFICATION	ALGO_XGBOOST
oml4sql-classification-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-classification-text-analysis-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-clustering-expectation-maximization.sql	CLUSTERING	ALGO_EXPECTATION_MAXIMIZATION
oml4sql-clustering-kmeanms-star-schema.sql	CLUSTERING	ALGO_KMEANS
oml4sql-clustering-kmeans.sql	CLUSTERING	ALGO_KMEANS
oml4sql-clustering-o-cluster.sql	CLUSTERING	ALGO_O_CLUSTER
oml4sql-partitioned-models-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-classification-regression-xgboost.sql	CLASSIFICATION	ALGO_XGBOOST
oml4sql-feature-extraction-text-analysis-esa.sql	FEATURE_EXTRACTION	ALGO_EXPLICIT_SEMANTIC_ANALYS
oml4sql-feature-extraction-nmf.sql	FEATURE_EXTRACTION	ALGO_NONNEGATIVE_MATRIX_FACTOR
oml4sql-feature-extraction-text-analysis-nmf.sql	FEATURE_EXTRACTION	ALGO_NONNEGATIVE_MATRIX_FACTOR
oml4sql-feature-extraction-svd.sql	FEATURE_EXTRACTION	ALGO_SINGULAR_VALUE_DECOMP
oml4sql-feature-extraction-text-mining-esa.sql	FEATURE_EXTRACTION	ALGO_EXPLICIT_SEMANTIC_ANALYSIS
oml4sql-regression-glm.sql	REGRESSION	ALGO_GENERALIZED_LINEAR_MODEL
oml4sql-regression-neural-networks.sql	REGRESSION	ALGO_NEURAL_NETWORK
oml4sql-regression-random-forest.sql	REGRESSION	ALGO_RANDOM_FOREST
oml4sql-regression-svm.sql	REGRESSION	ALGO_SUPPORT_VECTOR_MACHINES

Table 41-1 (Cont.) Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
oml4sql-singular-value-decomposition.sql	REGRESSION	ALGO_SINGULAR_VALUE_DECOMPOSITION
oml4sql-survival-analysis-xgboost.sql	REGRESSION	ALGO_XGBOOST
oml4sql-time-series-esm-auto-model-search.sql	TIME_SERIES	ALGO_EXPONENTIAL_SMOOTHING
oml4sql-time-series-exponential-smoothing.sql	TIME_SERIES	ALGO_EXPONENTIAL_SMOOTHING
oml4sql-time-series-regression-dataset.sql	-	This is a dataset to construct time series regression model.
oml4sql-time-series-regression.sql	TIME_SERIES and REGRESSION	Uses ALGO_EXPONENTIAL_SMOOTHING, ALGO_GENERALIZED_MODEL, and ALGO_XGBOOST

A few examples other than those listed in the table above are: `oml4sql-attribute-importance.sql`, which uses the `DBMS_PREDICTIVE_ANALYTICS.EXPLAIN` procedure to find the importance of attributes that independently impact the target attribute. `oml4sql-feature-extraction-text-term-extraction.sql` example, which uses the `CTX.DDL` package for text extraction.

Another set of examples demonstrates the use of the `ALGO_EXTENSIBLE_LANG` algorithm to register R language functions and create R models. The following table lists the R Extensibility examples. It shows the file name of the example and the MINING_FUNCTION value and R function used.

File Name	MINING_FUNCTION	R Function
oml4sql-r-extensible-algorithm-registration.sql	CLASSIFICATION	glm
oml4sql-r-extensible-association-rules.sql	ASSOCIATION	apriori
oml4sql-r-extensible-attribute-importance-via-rf.sql	REGRESSION	randomForest
oml4sql-r-extensible-glm.sql	REGRESSION	glm
oml4sql-r-extensible-kmeans.sql	CLUSTERING	kmeans
oml4sql-r-extensible-principal-components.sql	FEATURE_EXTRACTION	prcomp
oml4sql-r-extensible-regression-tree.sql	REGRESSION	rpart
oml4sql-r-extensible-regression-neural-networks.sql	REGRESSION	nnet

41.2 Install the OML4SQL Examples

Learn how to install OML4SQL examples.

The OML4SQL examples require:

- Oracle Database (on-premises, Oracle Database Cloud Service, or Oracle Autonomous Database)
- Oracle Database sample schemas
- A user account with the privileges described in Grant Privileges for Oracle Machine Learning for SQL.
- Running of `dmshgrants.sql` by a system administrator
- Running of `dmsh.sql` by the OML4SQL user

Follow these steps to install the OML4SQL examples:

1. Install or obtain access to an Oracle Database 23ai instance. To install the database, see the installation instructions for your platform at Oracle Database 23ai.
2. Ensure that the sample schemas are installed in the database. See *Oracle Database Sample Schemas* for details about the sample schemas.
3. Download the example code files from GitHub at https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql. Select the Database edition. Place the files in a directory to which you have access on the Oracle Database server. For example, `$ORACLE_HOME/demo/schema`. `$ORACLE_HOME` is the home path where you have installed the database. Typically, `/scratch/u01/app/oracle/product/23.0.0/dbhome_1`.
4. Verify that your user account has the required privileges described in Grant Privileges for Oracle Machine Learning for SQL.
5. Ask your system administrator to run the `dmshgrants.sql` script, or run it yourself if you have administrative privileges. The script grants the privileges that are required for running the examples. These include `SELECT` access to tables in the `SH` schema as described in OML4SQL Sample Data and the system privileges.

   Pass the name of the OML4SQL user to `dmshgrants`.

   ```
   SQL> CONNECT sys / as sysdba
   Enter password: sys_password
   Connected.
   SQL> @<location_of_examples>/dmshgrants oml_user
   ```
6. Connect to the database and run the `dmsh.sql` script. This script creates views of the sample data in the schema of the OML4SQL user.

   ```
   SQL> CONNECT oml_user
   Enter password: oml_user_password
   Connected.
   SQL> @<location_of_examples>/dmsh.sql
   ```

Related Topics

- *Oracle Database Sample Schemas*

41.3 OML4SQL Sample Data

The data used by the OML4SQL examples is based on these tables in the SH schema.

Those tables are:

```
SH.CUSTOMERS
SH.SALES
SH.PRODUCTS
SH.SUPPLEMENTARY_DEMOGRAPHICS
SH.COUNTRIES
```

The dmshgrants script grants SELECT access to the tables in the SH schema. The dmsh.sql script creates views of the SH tables in the schema of the OML4SQL user. The views are described in the following table.

Table 41-2 Views Created by dmsh.sql

View Name	Description
MINING_DATA	Joins and filters data
MINING_DATA_BUILD_V	Data for building models
MINING_DATA_TEST_V	Data for testing models
MINING_DATA_APPLY_V	Data to be scored
MINING_BUILD_TEXT	Data for building models that include text
MINING_TEST_TEXT	Data for testing models that include text
MINING_APPLY_TEXT	Data, including text columns, to be scored
MINING_DATA_ONE_CLASS_V	Data for anomaly detection

The association rules example creates its own transactional data.

Part V

Oracle Machine Learning for SQL API Reference

Learn about Oracle Machine Learning for SQL PL/SQL packages, data dictionary views, and machine learning SQL scoring functions.

42
PL/SQL Packages

Learn how to create, evaluate, and query machine learning models through Oracle Machine Learning for SQL PL/SQL packages.

- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_PREDICTIVE_ANALYTICS

42.1 DBMS_DATA_MINING

The `DBMS_DATA_MINING` package is the application programming interface for creating, evaluating, and querying Oracle Machine Learning for SQL models.

In Oracle Database Release 21c, Oracle Data Mining has been rebranded to Oracle Machine Learning for SQL (OML4SQL). The PL/SQL package name, however, has not changed and remains `DBMS_DATA_MINING`.

This chapter contains the following topics:

- Overview
- Security Model
- Mining Functions
- Model Settings
- Algorithm Specific Settings
- Solver Settings
- Datatypes
- Summary of DBMS_DATA_MINING Subprograms

> **See Also:**
>
> - *Oracle Machine Learning for SQL Concepts*
> - *Oracle Machine Learning for SQL User's Guide*
> - DBMS_DATA_MINING_TRANSFORM
> - DBMS_PREDICTIVE_ANALYTICS

42.1.1 DBMS_DATA_MINING Overview

Oracle Machine Learning for SQL supports both supervised and unsupervised machine learning. Supervised machine learning predicts a target value based on historical data.

Unsupervised machine learning discovers natural groupings and does not use a target. You can use OML4SQL procedures on structured data and unstructured text.

Supervised machine learning techniques include:

- Classification
- Regression
- Feature Selection (Attribute Importance)
- Time Series

Unsupervised machine learning techniques include:

- Clustering
- Association
- Feature Extraction
- Anomaly Detection

The steps you use to build and apply a machine learning model depend on the machine learning technique and the algorithm being used. The algorithms supported by Oracle Machine Learning for SQL are listed in the following table.

Table 42-1 OML4SQL Algorithms

Algorithm	Abbreviation	Function
Apriori	AR	Association
CUR Matrix Decomposition	CUR	Attribute importance
Decision Tree	DT	Classification
Expectation Maximization	EM	Clustering
Explicit Semantic Analysis	ESA	Feature extraction, classification
Exponential Smoothing	ESM	Time series
Generalized Linear Models	GLM	Classification, regression
k-Means	KM	Clustering
Minimum Descriptor Length	MDL	Attribute importance
Multivariate State Estimation Technique - Sequential Probability Ratio Test	MSET-SPRT	Anomaly detection, classification
Naive Bayes	NB	Classification
Neural Network	NN	Classification, regression
Non-Negative Matrix Factorization	NMF	Feature extraction
Orthogonal Partitioning Clustering	O-Cluster	Clustering
Random Forest	RF	Classification
Singular Value Decomposition and Principal Component Analysis	SVD and PCA	Feature extraction
Support Vector Machine	SVM	Classification, regression, anomaly detection
XGBoost	XGBoost	Classification, regression

OML4SQL supports more than one algorithm for the classification, regression, clustering, and feature extraction machine learning techniques. Each of these machine learning techniques has a default algorithm, as shown in the following table.

Table 42-2 OML4SQL Default Algorithms

Mining Function	Default Algorithm
Classification	Naive Bayes
Clustering	*k*-Means
Feature Extraction	Non-Negative Matrix Factorization
Feature Selection	Minimum Descriptor Length
Regression	Support Vector Machine
Time Series	Exponential Smoothing

42.1.2 DBMS_DATA_MINING Security Model

The `DBMS_DATA_MINING` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The `DBMS_DATA_MINING` package exposes APIs that are leveraged by the Oracle Machine Learning for SQL. Users who wish to create machine learning models in their own schema require the `CREATE MINING MODEL` system privilege. Users who wish to create machine learning models in other schemas require the `CREATE ANY MINING MODEL` system privilege.

Users have full control over managing models that exist within their own schema. Additional system privileges necessary for managing machine learning models in other schemas include `ALTER ANY MINING MODEL`, `DROP ANY MINING MODEL`, `SELECT ANY MINING MODEL`, `COMMENT ANY MINING MODEL`, and `AUDIT ANY`.

Individual object privileges on machine learning models, `ALTER MINING MODEL` and `SELET MINING MODEL`, can be used to selectively grant privileges on a model to a different user.

> **See Also:**
>
> *Oracle Data Mining User's Guide* for more information about the security features of OML4SQL

42.1.3 DBMS_DATA_MINING — Machine Learning Functions

A machine learning **function** refers to the methods for solving a given class of machine learning problems.

The machine learning function must be specified when a model is created. You specify a machine learning function with the `mining_function` parameter of the CREATE_MODEL Procedure or the CREATE_MODEL2 Procedure.

Table 42-3 Machine Learning Functions

Value	Description
`ASSOCIATION`	Association is a descriptive machine learning function. An association model identifies relationships and the probability of their occurrence within a data set. Association models use the Apriori algorithm.
`ATTRIBUTE_IMPORTANCE`	Attribute importance is a predictive machine learning function, also known as feature selection. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome. Attribute importance models can use Minimum Description Length (MDL) or CUR Matrix Decomposition. MDL is the default.
`CLASSIFICATION`	Classification is a predictive machine learning function. A classification model uses historical data to predict a categorical target. Classification models can use: Decision Tree, logistic regression, Multivariate State Estimation Technique - Sequential Probability Ratio Test, Naive Bayes, Support Vector Machine (SVM), or XGBoost. The default is Naive Bayes. The classification function can also be used for **anomaly detection**. For anomaly detection, you can use the Multivariate State Estimation Technique - Sequential Probability Ratio Test algorithm or the SVM algorithm with a null target (One-Class SVM), or the EM algorithm with a null target (EM Anomaly).
`CLUSTERING`	Clustering is a descriptive machine learning function. A clustering model identifies natural groupings within a data set. Clustering models can use *k*-Means, O-Cluster, or Expectation Maximization. The default is *k*-Means.
`FEATURE_EXTRACTION`	Feature extraction is a descriptive machine learning function. A feature extraction model creates an optimized data set on which to base a model. Feature extraction models can use Explicit Semantic Analysis, Non-Negative Matrix Factorization, Singular Value Decomposition, or Principal Component Analysis. Non-Negative Matrix Factorization is the default.
`REGRESSION`	Regression is a predictive machine learning function. A regression model uses historical data to predict a numerical target. Regression models can use linear regression, Support Vector Machine, or XGBoost. The default is Support Vector Machine.
`TIME_SERIES`	Time series is a predictive machine learning function. A time series model forecasts the future values of a time-ordered series of historical numeric data over a user-specified time window. Time series models use the Exponential Smoothing algorithm.

See Also:

Oracle Machine Learning for SQL Concepts for more information about mining functions

42.1.4 DBMS_DATA_MINING — Model Settings

Oracle Machine Learning for SQL uses settings to specify the algorithm and other characteristics of a model. Some settings are general, some are specific to a machine learning function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, then you must create a settings table. The settings table must have the column names and data types shown in the following table.

Table 42-4 Required Columns in the Model Settings Table

Column Name	Data Type
`SETTING_NAME`	`VARCHAR2(30)`
`SETTING_VALUE`	`VARCHAR2(4000)`

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the CREATE_MODEL Procedure. You can also provide these settings through the CREATE_MODEL2 Procedure.

The settings used by a model can be found by querying the data dictionary view `ALL_MINING_MODEL_SETTINGS`. This view displays the model settings used by the machine learning models to which you have access. All of the default and user-specified setting values are included in the view.

> **See Also:**
>
> - `ALL_MINING_MODEL_SETTINGS` in *Oracle Database Reference*
> - *Oracle Machine Learning for SQL User's Guide* for information about specifying model settings

42.1.4.1 DBMS_DATA_MINING — Algorithm Names

The `ALGO_NAME` setting specifies the model algorithm.

The values for the `ALGO_NAME` setting are listed in the following table.

Table 42-5 Algorithm Names

ALGO_NAME Value	Description	Machine Learning Function
`ALGO_AI_MDL`	Minimum Description Length	Attribute importance
`ALGO_APRIORI_ASSOCIATION_RULES`	Apriori	Association rules
`ALGO_CUR_DECOMPOSITION`	CUR Matrix Decomposition	Attribute importance
`ALGO_DECISION_TREE`	Decision Tree	Classification
`ALGO_EXPECTATION_MAXIMIZATION`	Expectation Maximization	Clustering, Classification

Table 42-5 (Cont.) Algorithm Names

ALGO_NAME Value	Description	Machine Learning Function
`ALGO_EXPLICIT_SEMANTIC_ANALYS`	Explicit Semantic Analysis	Feature extraction Classification
`ALGO_EXPONENTIAL_SMOOTHING`	Exponential Smoothing	Time series
`ALGO_EXTENSIBLE_LANG`	Language used for extensible algorithm	All mining functions supported
`ALGO_GENERALIZED_LINEAR_MODEL`	Generalized Linear Model	Classification, regression; also feature selection and generation
`ALGO_KMEANS`	Enhanced *k*-Means	Clustering
`ALGO_MSET_SPRT`	Multivariate State Estimation Technique - Sequential Probability Ratio Test	Classification
`ALGO_NAIVE_BAYES`	Naive Bayes	Classification
`ALGO_NEURAL_NETWORK`	Neural Network	Classification
`ALGO_NONNEGATIVE_MATRIX_FACTOR`	Non-Negative Matrix Factorization	Feature extraction
`ALGO_O_CLUSTER`	O-Cluster	Clustering
`ALGO_RANDOM_FOREST`	Random Forest	Classification
`ALGO_SINGULAR_VALUE_DECOMP`	Singular Value Decomposition	Feature extraction
`ALGO_SUPPORT_VECTOR_MACHINES`	Support Vector Machine	Classification and regression
`ALGO_XGBOOST`	XGBoost	Classification and regression

See Also:

Oracle Machine Learning for SQL Concepts for information about algorithms

42.1.4.2 DBMS_DATA_MINING — Automatic Data Preparation

Oracle Machine Learning for SQL supports fully Automatic Data Preparation (ADP), user-directed general data preparation, and user-specified embedded data preparation. The `PREP_*` settings enable the user to request fully automated or user-directed general data preparation. By default, fully Automatic Data Preparation (`PREP_AUTO_ON`) is enabled.

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. Instead of fully ADP, the user can request that the data be shifted and/or scaled with the `PREP_SCALE*` and `PREP_SHIFT*` settings. The transformation instructions are stored with the model and reused whenever the model is applied. The model settings can be viewed in `USER_MINING_MODEL_SETTINGS`.

You can choose to supplement Automatic Data Preparations by specifying additional transformations in the `xform_list` parameter when you build the model. See "CREATE_MODEL Procedure" and "CREATE_MODEL2 Procedure".

If you do not use ADP *and* do not specify transformations in the `xform_list` parameter to `CREATE_MODEL`, you must implement your own transformations separately in the build, test, and scoring data. You must take special care to implement the exact same transformations in each data set.

If you do not use ADP, but you *do* specify transformations in the `xform_list` parameter to `CREATE_MODEL`, OML4SQL embeds the transformation definitions in the model and prepares the test and scoring data to match the build data.

The values for the `PREP_*` setting are described in the following table.

Table 42-6 PREP_* Setting

Setting Name	Setting Value	Description
`PREP_AUTO`	• `PREP_AUTO_ON` • `PREP_AUTO_OFF`	This setting enables fully automated data preparation. The default is `PREP_AUTO_ON`.
`PREP_SCALE_2DNUM`	• `PREP_SCALE_STDDEV` • `PREP_SCALE_RANGE`	This setting enables scaling data preparation for two-dimensional numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. The following are the possible values: • `PREP_SCALE_STDDEV`: A request to divide the column values by the standard deviation of the column and is often provided together with `PREP_SHIFT_MEAN` to yield z-score normalization. • `PREP_SCALE_RANGE`: A request to divide the column values by the range of values and is often provided together with `PREP_SHIFT_MIN` to yield a range of [0,1].
`PREP_SCALE_NNUM`	`PREP_SCALE_MAXABS`	This setting enables scaling data preparation for nested numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. If specified, then the valid value for this setting is `PREP_SCALE_MAXABS`, which yields data in the range of [-1,1].
`PREP_SHIFT_2DNUM`	• `PREP_SHIFT_MEAN` • `PREP_SHIFT_MIN`	This setting enables centering data preparation for two-dimensional numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. The following are the possible values: • `PREP_SHIFT_MEAN`: Results in subtracting the average of the column from each value. • `PREP_SHIFT_MIN`: Results in subtracting the minimum of the column from each value. .

See Also:

Oracle® Machine Learning for SQL for information about data transformations

42.1.4.3 DBMS_DATA_MINING — Machine Learning Function Settings

The settings described in this table apply to a machine learning function.

Table 42-7 Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Association	`ASSO_MAX_RULE_LENGTH`	`TO_CHAR( 2<= numeric_expr <=20)`	Maximum rule length for association rules. Default is `4`.
Association	`ASSO_MIN_CONFIDENCE`	`TO_CHAR( 0<= numeric_expr <=1)`	Minimum confidence for association rules. Default is `0.1`.
Association	`ASSO_MIN_SUPPORT`	`TO_CHAR( 0<= numeric_expr <=1)`	Minimum support for association rules Default is `0.1`.
Association	`ASSO_MIN_SUPPORT_INT`	a positive integer	Minimum absolute support that each rule must satisfy. The value must be an integer. The default is `1`.
Association	`ASSO_MIN_REV_CONFIDENCE`	`TO_CHAR( 0<= numeric_expr <=1)`	Sets the Minimum Reverse Confidence that each rule should satisfy. The Reverse Confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs. The value is real number between 0 and 1. The default is `0`.

Association	`ASSO_IN_RULES`	`NULL`	Sets Including Rules applied for each association rule: it specifies the list of items that at least one of them must appear in each reported association rule, either as antecedent or as consequent. It is a comma separated string containing the list of including items.
			If not set, the default behavior is, the filtering is not applied.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_in_rules, '''a'',''b''');`

Table 42-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Association	`ASSO_EX_RULES`	`NULL`	Sets Excluding Rules applied for each association rule: it specifies the list of items that none of them can appear in each reported association rules. It is a comma separated string containing the list of excluded items. No rule can contain any item in the list.
			The default is `NULL`.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_ex_rules, '''a'',''b''');`

Association	`ASSO_ANT_IN_RULES`	`NULL`	Sets Including Rules for the antecedent: it specifies the list of items that at least one of them must appear in the antecedent part of each reported association rule. It is a comma separated string containing the list of including items. The antecedent part of each rule must contain at least one item in the list.
			The default is `NULL`.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_ant_in_rules, '''a'',''b''');`

Association	`ASSO_ANT_EX_RULES`	`NULL`	Sets Excluding Rules for the antecedent: it specifies the list of items that none of them can appear in the antecedent part of each reported association rule. It is a comma separated string containing the list of excluded items. No rule can contain any item in the list in its antecedent part.
			The default is `NULL`.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_ant_ex_rules, '''a'',''b''');`

Table 42-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Association	`ASSO_CONS_IN_RULES`	`NULL`	Sets Including Rules for the consequent: it specifies the list of items that at least one of them must appear in the consequent part of each reported association rule. It is a comma separated string containing the list of including items. The consequent of each rule must be an item in the list.
			The default is `NULL`.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_cons_in_rules , '''a'',''b''');`

Association	`ASSO_CONS_EX_RULES`	`NULL`	Sets Excluding Rules for the consequent: it specifies the list of items that none of them can appear in the consequent part of each reported association rule. It is a comma separated string containing the list of excluded items. No rule can have any item in the list as its consequent.
			The excluding rule can be used to reduce the data that must be stored, but the user may be required to build an extra model for executing different including or Excluding Rules.
			The default is `NULL`.
			For example,
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_cons_ex_rules , '''a'',''b''');`

Table 42-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Association	`ASSO_AGGREGATES`	`NULL`	Specifies the columns to be aggregated. It is a comma separated string containing the names of the columns for aggregation. The number of columns in the list must be <= 10. You can set `ASSO_AGGREGATES` if `ODMS_ITEM_ID_COLUMN_NAME` is set indicating transactional input data. See DBMS_DATA_MINING - Global Settings. The data table must have valid column names such as `ITEM_ID` and `CASE_ID` which are derived from `ODMS_ITEM_ID_COLUMN_NAME` and `case_id_column_name` respectively. Numeric values are supported. `ITEM_VALUE` is not a mandatory value. The default is `NULL`. For each item, the user may supply several columns to aggregate. It requires more memory to buffer the extra data. Also, the performance impact can be seen because of the larger input data set and more operation.
Association	`ASSO_ABS_ERROR`	`0<ASSO_ABS_ERRORMAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE).`	Specifies the absolute error for the association rules sampling. A smaller value of `ASSO_ABS_ERROR` obtains a larger sample size which gives accurate results but takes longer computational time. Set a reasonable value for `ASSO_ABS_ERROR`, such as its default value, to avoid large sample size. The default value is `0.5` * `MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)`.
Association	`ASSO_CONF_LEVEL`	`0 ASSO_CONF_LEVEL 1`	Specifies the confidence level for an association rules sample. A larger value of `ASSO_CONF_LEVEL` obtains a larger sample size. Any value between `0.9` and `1` is suitable. The default value is `0.95`.
Classification	`CLAS_COST_TABLE_NAME`	*table_name*	(Decision tree only) Name of a table that stores a cost matrix to be used by the algorithm in building the model. The cost matrix specifies the costs associated with misclassifications. Only decision tree models can use a cost matrix at build time. All classification algorithms can use a cost matrix at apply time. The cost matrix table is user-created. See "ADD_COST_MATRIX Procedure" for the column requirements. See *Oracle Machine Learning for SQL Concepts* for information about costs.

Table 42-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Classification	`CLAS_PRIORS_TABLE_NAME`	*table_name*	(Naive Bayes) Name of a table that stores prior probabilities to offset differences in distribution between the build data and the scoring data. The priors table is user-created. See *Oracle Machine Learning for SQL User's Guide* for the column requirements. See *Oracle Machine Learning for SQL Concepts* for additional information about priors.
Classification	`CLAS_WEIGHTS_TABLE_NAME`	*table_name*	(GLM and SVM only) Name of a table that stores weighting information for individual target values in SVM classification and GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes. The class weights table is user-created. See *Oracle Machine Learning for SQL User's Guide* for the column requirements. See *Oracle Machine Learning for SQL Concepts* for additional information about class weights.
Classification	`CLAS_WEIGHTS_BALANCED`	`ON` `OFF`	This setting indicates that the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is `OFF`.

Classification	`CLAS_MAX_SUP_BINS`	For Decision Tree:
		`2 <= a number <=2147483647`
		For Random Forest:
		`2 <= a number <=254`
			This parameter specifies the maximum number of bins for each attribute.
			The default value is `32`.
			See, DBMS_DATA_MINING — Automatic Data Preparation

Table 42-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Clustering	`CLUS_NUM_CLUSTERS`	*TO_CHAR(numeric_expr* >=1)	The maximum number of leaf clusters generated by a clustering algorithm. The algorithm may return fewer clusters, depending on the data. Enhanced *k*-Means usually produces the exact number of clusters specified by `CLUS_NUM_CLUSTERS`, unless there are fewer distinct data points. When Expectation maximization (EM) is used for clustering, it may return fewer clusters than the number specified by `CLUS_NUM_CLUSTERS` depending on the data. The number of clusters returned by EM cannot be greater than the number of components, which is governed by algorithm-specific settings. (See *Expectation Maximization Settings for Learning* table) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, the number of clusters equals the number of components. The setting can be used only for EM Clustering algorithm. For EM Clustering algorithm, the default value of `CLUS_NUM_CLUSTERS` is system-determined. For *k*-Means and O-Cluster, the default is `10`.
Feature extraction	`FEAT_NUM_FEATURES`	TO_CHAR(*numeric_expr* >=1)	The number of features to be extracted by a feature extraction model. The default is estimated from the data by the algorithm. If the matrix rank is smaller than this number, fewer features will be returned. For CUR Matrix Decomposition, the `FEAT_NUM_FEATURES` value is the same as the `CURS_SVD_RANK` value.

See Also:

Oracle Machine Learning for SQL Concepts for information about machine learning functions

42.1.4.4 DBMS_DATA_MINING — Global Settings

The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

Table 42-8 Global Settings

Setting Name	Setting Value	Description
`ODMS_BOXCOX`	`ODMS_BOXCOX_ENABLE` `ODMS_BOXCOX_DISABLE`	This setting enables the Box-Cox variance-stabilization transformation. It is useful when the variance increases as the target value increases. It reduces variance and transforms a multiplicative relationship with the target, with a simpler additive relationship. This setting is applicable only to the Exponential Smoothing algorithm. When a value for `EXSM_MODEL` setting is not specified, the default value is `ODMS_BOXCOX_ENABLE` and when a value for the `EXSM_MODEL` setting is provided, the default value is `ODMS_BOXCOX_DISABLE`.
`ODMS_EXPLOSION_MIN_SUPP`	A positive integer	It is the minimum required support for categorical values that must be included in the explosion mapping. It removes categorical values with insufficient row instances to have a statistically significant effect on the model, however, they could potentially degrade performance. The default is system determined depending on the number of rows in the dataset. A value of `1` results into mapping all categorical values.
`ODMS_ITEM_ID_COLUMN_NAME`	*column_name*	(Association rules only) Name of a column that contains the items in a transaction. When this setting is specified, the algorithm expects the data to be presented in a native transactional format, consisting of two columns: • Case ID, either categorical or numeric • Item ID, either categorical or numeric **Note:** Oracle Machine Learning does not support `BOOLEAN` values for this setting. A typical example of transactional data is market basket data, wherein a case represents a basket that may contain many items. Each item is stored in a separate row, and many rows may be needed to represent a case. The case ID values do not uniquely identify each row. Transactional data is also called multi-record case data. Association rules function is normally used with transactional data, but it can also be applied to single-record case data (similar to other algorithms). For more information about single-record and multi-record case data, see *Oracle SQL Developer Data Modeler User's Guide*.

Table 42-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_ITEM_VALUE_COLUMN_NAME	*column_name*	(Association rules only) Name of a column that contains a value associated with each item in a transaction. This setting is only used when a value has been specified for ODMS_ITEM_ID_COLUMN_NAME indicating that the data is presented in native transactional format. If ASSO_AGGREGATES is used, then the build data must include the following three columns and the columns specified in the AGGREGATES setting. • Case ID, either categorical or numeric • Item ID, either categorical or numeric, specified by ODMS_ITEM_ID_COLUMN_NAME • Item value, either categorical or numeric, specified by ODMS_ITEM_VALUE_COLUMN_NAME **Note:** Oracle Machine Learning does not support BOOLEAN values for this setting. If ASSO_AGGREGATES, Case ID, and Item ID column are present, then the Item Value column may or may not appear. The Item Value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples). For details on ASSO_AGGREGATES, see DBMS_DATA_MINING - Mining Function Settings.
ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_MEAN_MODE ODMS_MISSING_VALUE_DELETE_ROW ODMS_MISSING_VALUE_AUTO	Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO. ODMS_MISSING_VALUE_MEAN_MODE replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time where appropriate. ODMS_MISSING_VALUE_AUTO performs different strategies for different algorithms. When ODMS_MISSING_VALUE_TREATMENT is set to ODMS_MISSING_VALUE_DELETE_ROW, the rows in the training data that contain missing values are deleted. However, if you want to replicate this missing value treatment in the scoring data, then you must perform the transformation explicitly. The value ODMS_MISSING_VALUE_DELETE_ROW applies to all algorithms.

Table 42-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_ROW_WEIGHT_COLUMN_NAME	*column_name*	(GLM only) Name of a column in the training data that contains a weighting factor for the rows. The column data type must be numeric. Oracle Machine Learning does not support BOOLEAN values for this setting. Row weights can be used as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. Row weights can also be used to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.
ODMS_TEXT_POLICY_NAME	The name of an Oracle Text POLICY created using CTX_DDL.CREATE_POLICY.	Affects how individual tokens are extracted from unstructured text. For details about CTX_DDL.CREATE_POLICY, see *Oracle Text Reference*.
ODMS_TEXT_MAX_FEATURES	1 <= *value*	The maximum number of distinct features, across all text attributes, to use from a document set passed to CREATE_MODEL. The default is 3000. ESA has the default value of 300000.
ODMS_TEXT_MIN_DOCUMENTS	Non-negative value	This is a text processing setting the controls how in how many documents a token needs to appear to be used as a feature. The default is 1. ESA has a default of 3.
ODMS_PARTITION_COLUMNS	Comma separated list of machine learning attributes	This setting indicates a request to build a partitioned model. The setting value is a comma-separated list of the machine learning attributes used to determine the in-list partition key values. Oracle Machine Learning supports numeric and categorical values including BOOLEAN for this setting. These machine learning attributes are taken from the input columns unless an XFORM_LIST parameter is passed to CREATE_MODEL or CREATE_MODEL2. If the XFORM_LIST parameter is passed to during model building, then the machine learning attributes are taken from the attributes produced by these transformations.
ODMS_MAX_PARTITIONS	1< value <= 1000000	This setting indicates the maximum number of partitions allowed for the model. The default is 1000.
ODMS_SAMPLING	ODMS_SAMPLING_ENABLE ODMS_SAMPLING_DISABLE	This setting allows the user to request a sampling of the build data. The default is ODMS_SAMPLING_DISABLE.
ODMS_SAMPLE_SIZE	0 < Value	This setting determines how many rows will be sampled (approximately). It can be set only if ODMS_SAMPLING is enabled. The default value is the system determined.

Table 42-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_PARTITION_BUILD_TYPE	ODMS_PARTITION_BUILD_INTRA ODMS_PARTITION_BUILD_INTER ODMS_PARTITION_BUILD_HYBRID	This setting controls the parallel build of partitioned models. ODMS_PARTITION_BUILD_INTRA — Each partition is built in parallel using all replicas. ODMS_PARTITION_BUILD_INTER — Each partition is built entirely in a single slave, but multiple partitions may be built at the same time since multiple replicas are active. ODMS_PARTITION_BUILD_HYBRID — It is a combination of the other two types and is recommended for most situations to adapt to dynamic environments. The default mode is ODMS_PARTITION_BUILD_HYBRID
ODMS_TABLESPACE_NAME	*tablespace_name*	This setting controls the storage specifications. If you explicitly sets this to the name of a tablespace (for which you have sufficient quota), then the specified tablespace storage creates the resulting model content. If you do not provide this setting, then the default tablespace of the user creates the resulting model content.
ODMS_RANDOM_SEED	The value must be a non-negative integer	The hash function with a random number seed generates a random number with uniform distribution. Users can control the random number seed by this setting. The default is 0. This setting is used by Random Forest, Neural Network, and CUR Matrix Decomposition.
ODMS_DETAILS	• ODMS_ENABLE • ODMS_DISABLE	This setting reduces the space that is used while creating a model, especially a partitioned model. The default value is ODMS_ENABLE. When the setting is ODMS_ENABLE, it creates model tables and views when the model is created. You can query the model with SQL. When the setting is ODMS_DISABLE, model views are not created and tables relevant to model details are not created either. The reduction in space depends on the model. Reduction on the order of 10x can be achieved.

See Also:

Oracle Machine Learning for SQL Concepts for information about GLM

Oracle Machine Learning for SQL Concepts for information about association rules

Oracle Machine Learning for SQL User's Guide for information about machine learning unstructured text

42.1.5 DBMS_DATA_MINING — Algorithm Specific Model Settings

Oracle Machine Learning for SQL uses algorithm specific settings to define the characteristics of a model.

All settings have default values. If you want to override one or more of the settings for a model, then you must specify those settings.

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the CREATE_MODEL Procedure. You can also provide these settings through the CREATE_MODEL2 Procedure.

The settings used by a model can be found by querying the data dictionary view `ALL_MINING_MODEL_SETTINGS`. This view displays the model settings used by the machine learning models to which you have access. All of the default and user-specified setting values are included in the view.

> **See Also:**
>
> - `ALL_MINING_MODEL_SETTINGS` in *Oracle Database Reference*
> - *Oracle Machine Learning for SQL User's Guide* for information about specifying model settings

42.1.5.1 DBMS_DATA_MINING — Algorithm Settings: ALGO_EXTENSIBLE_LANG

The settings listed in the following table configure the behavior of the machine learning model with an extensible algorithm. The model is built in the R language.

The `RALG_*_FUNCTION` specifies the R script that is used to build, score, and view an R model and must be registered in the Oracle Machine Learning for R script repository. The R scripts are registered through OML4R with special privileges. When `ALGO_EXTENSIBLE_LANG` is set to R in the `MINING_MODEL_SETTING` table, the machine learning model is built in the R language. After the R model is built, the names of the R scripts are recorded in the `MINING_MODEL_SETTING` table in the `SYS` schema. The scripts must exist in the script repository for the R model to function. The amount of R memory used to build, score, and view the R model through these R scripts can be controlled by OML4R.

All algorithm-independent `DBMS_DATA_MINING` subprograms can operate on an R model for machine learning functions such as association, attribute importance, classification, clustering, feature extraction, and regression.

The supported `DBMS_DATA_MINING` subprograms include, but are not limited, to the following:

- ADD_COST_MATRIX Procedure
- COMPUTE_CONFUSION_MATRIX Procedure
- COMPUTE_LIFT Procedure

- COMPUTE_ROC Procedure
- CREATE_MODEL Procedure
- DROP_MODEL Procedure
- EXPORT_MODEL Procedure
- GET_MODEL_COST_MATRIX Function
- IMPORT_MODEL Procedure
- REMOVE_COST_MATRIX Procedure
- RENAME_MODEL Procedure

Table 42-9 ALGO_EXTENSIBLE_LANG Settings

Setting Name	Setting Value	Description
`RALG_BUILD_FUNCTION`	`R_BUILD_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm machine learning model build function. The R script defines an R function for the first input argument for training data and returns an R model object. For clustering and feature extraction machine learning function model build, the R attributes `dm$nclus` and `dm$nfeat` must be set on the R model to indicate the number of clusters and features respectively. The `RALG_BUILD_FUNCTION` must be set along with `ALGO_EXTENSIBLE_LANG` in the `model_setting_table`.
`RALG_BUILD_PARAMETER`	`SELECT` *value* `param_name`, ...`FROM DUAL`	Specifies a list of numeric and string scalar for optional input parameters of the model build function.
`RALG_SCORE_FUNCTION`	`R_SCORE_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script to score data. The script returns a `data.frame` containing the corresponding prediction results. The setting is used to score data for machine learning functions such as regression, classification, clustering, and feature extraction. This setting does not apply to the association and the attribute importance functions.
`RALG_WEIGHT_FUNCTION`	`R_WEIGHT_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm that computes the weight (contribution) for each attribute in scoring. The script returns a `data.frame` containing the contributing weight for each attribute in a row. This function setting is needed for the `PREDICTION_DETAILS` SQL function.
`RALG_DETAILS_FUNCTION`	`R_DETAILS_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm that produces the model information. This setting is required to generate a model view.

Table 42-9 (Cont.) ALGO_EXTENSIBLE_LANG Settings

Setting Name	Setting Value	Description
RALG_DETAILS_FORMAT	SELECT *type_value column_name*, ... FROM DUAL	Specifies the SELECT query for the list of numeric and string scalars for the output column type and the column name of the generated model view. This setting is required to generate a model view.

See Also:

Oracle Machine Learning for SQL User's Guide

42.1.5.2 DBMS_DATA_MINING — Algorithm Settings: CUR Matrix Decomposition

The following settings affects the behavior of the CUR Matrix Decomposition algorithm.

Table 42-10 CUR Matrix Decomposition Settings

Setting Name	Setting Value	Description
CURS_APPROX_ATTR_NUM	The value must be a positive integer	Defines the approximate number of attributes to be selected. The default value is the number of attributes.
CURS_ROW_IMPORTANCE	CURS_ROW_IMP_ENABLE CURS_ROW_IMP_DISABLE	Defines the flag indicating whether or not to perform row selection. The default value is CURS_ROW_IMP_DISABLE.
CURS_APPROX_ROW_NUM	The value must be a positive integer	Defines the approximate number of rows to be selected. This parameter is only used when users decide to perform row selection (CURS_ROW_IMP_ENABLE). The default value is the total number of rows.
CURS_SVD_RANK	The value must be a positive integer	Defines the rank parameter used in the column/row leverage score calculation. If users do not provide an input value, the value is determined by the system.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts

42.1.5.3 DBMS_DATA_MINING — Algorithm Settings: Decision Tree

These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual decision tree.

Table 42-11 Decision Tree Settings

Setting Name	Setting Value	Description
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	Tree impurity metric for Decision Tree. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split values are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is by a metric. Decision trees can use either Gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.
TREE_TERM_MAX_DEPTH	For Decision Tree: 2<= *a number* <=20 For Random Forest: 2<= *a number* <=100	Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). For Decision Tree, the default is 7. For Random Forest, the default is 16.
TREE_TERM_MINPCT_NODE	0<= *a number*<=10	The minimum number of training rows in a node expressed as a percentage of the rows in the training data. Default is 0.05, indicating 0.05%.
TREE_TERM_MINPCT_SPLIT	0 < *a number* <=20	The minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. Default is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	*a number*>=0	The minimum number of rows in a node. Default is 10.
TREE_TERM_MINREC_SPLIT	a number > 1	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. Default is 20.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.

- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about Decision Tree

42.1.5.4 DBMS_DATA_MINING — Algorithm Settings: Expectation Maximization

These algorithm settings configure the behavior of the Expectation Maximization algorithm.

> **See Also:**
>
> *Oracle Data Mining Concepts* for information about Expectation Maximization

Table 42-12 Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Setting Value	Description
`EMCS_ATTRIBUTE_FILTER`	`EMCS_ATTR_FILTER_ENABLE` `EMCS_ATTR_FILTER_DISABLE`	Whether or not to include uncorrelated attributes in the model. When `EMCS_ATTRIBUTE_FILTER` is enabled, uncorrelated attributes are not included. **Note:** This setting applies only to attributes that are not nested. For Clustering, the default is system-determined. For anomaly detection, the default is `EMCS_ATTR_FILTER_DISABLE`.
`EMCS_MAX_NUM_ATTR_2D`	`TO_CHAR(` *`numeric_expr`* `>=1)`	Maximum number of correlated attributes to include in the model. Note: This setting applies only to attributes that are not nested (2D). Default is `50`.

Table 42-12 (Cont.) Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Setting Value	Description
EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_BERNOULLI EMCS_NUM_DISTR_GAUSSIAN EMCS_NUM_DISTR_SYSTEM	The distribution for modeling numeric attributes. Applies to the input table or view as a whole and does not allow per-attribute specifications. The options include Bernoulli, Gaussian, or system-determined distribution. When Bernoulli or Gaussian distribution is chosen, all numeric attributes are modeled using the same type of distribution. When the distribution is system-determined, individual attributes may use different distributions (either Bernoulli or Gaussian), depending on the data. Default is `EMCS_NUM_DISTR_SYSTEM`.
EMCS_NUM_EQUIWIDTH_BINS	TO_CHAR(1 <*numeric_expr* <=255)	Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. Default is `11`.
EMCS_NUM_PROJECTIONS	TO_CHAR(*numeric_expr* >=1)	Specifies the number of projections that will be used for each nested column. If a column has fewer distinct attributes than the specified number of projections, the data will not be projected. The setting applies to all nested columns. Default is `50`.
EMCS_NUM_QUANTILE_BINS	TO_CHAR(1 <*numeric_expr* <=255)	Specifies the number of quantile bins that will be used for modeling numeric columns with multivalued Bernoulli distributions. Default is system-determined.
EMCS_NUM_TOPN_BINS	TO_CHAR(1 <*numeric_expr* <=255)	Specifies the number of top-N bins that will be used for modeling categorical columns with multivalued Bernoulli distributions. Default is system-determined.

Table 42-13 Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_HELDASIDE EMCS_CONV_CRIT_BIC	The convergence criterion for EM. The convergence criterion may be based on a held-aside data set, or it may be Bayesian Information Criterion. Default is system determined.
EMCS_LOGLIKE_IMPROVEMENT	TO_CHAR(0 < *numeric_expr* < 1)	When the convergence criterion is based on a held-aside data set (`EMCS_CONVERGENCE_CRITERION` = `EMCS_CONV_CRIT_HELDASIDE`), this setting specifies the percentage improvement in the value of the log likelihood function that is required for adding a new component to the model. Default value is `0.001`.

Table 42-13 (Cont.) Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_NUM_COMPONENTS	TO_CHAR(*numeric_expr* >=1)	Maximum number of components in the model. If model search is enabled, the algorithm automatically determines the number of components based on improvements in the likelihood function or based on regularization, up to the specified maximum. For EM Clustering, the number of components must be greater than or equal to the number of clusters. Default is 20 for both EM Clustering and EM Anomaly.
EMCS_NUM_ITERATIONS	TO_CHAR(*numeric_expr* >=1)	Specifies the maximum number of iterations in the EM algorithm. Default is 100.
EMCS_MODEL_SEARCH	EMCS_MODEL_SEARCH_ENABLE EMCS_MODEL_SEARCH_DISABLE (default).	This setting enables model search in EM where different model sizes are explored and a best size is selected. The default is EMCS_MODEL_SEARCH_DISABLE.
EMCS_REMOVE_COMPONENTS	EMCS_REMOVE_COMPS_ENABLE (default) EMCS_REMOVE_COMPS_DISABLE	This setting allows the EM algorithm to remove a small component from the solution. The default is EMCS_REMOVE_COMPS_ENABLE.
EMCS_RANDOM_SEED	Non-negative integer	This setting controls the seed of the random generator used in EM. The default is 0.

Table 42-14 Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
EMCS_CLUSTER_COMPONENTS	EMCS_CLUSTER_COMP_ENABLE EMCS_CLUSTER_COMP_DISABLE	Enables or disables the grouping of EM components into high-level clusters. When disabled, the components themselves are treated as clusters. The setting can be used only for EM Clustering. When component clustering is enabled, model scoring through the SQL CLUSTER function will produce assignments to the higher level clusters. When clustering is disabled, the CLUSTER function will produce assignments to the original components. Default is EMCS_CLUSTER_COMP_ENABLE.
EMCS_CLUSTER_THRESH	TO_CHAR(*numeric_expr* >=1)	Dissimilarity threshold that controls the clustering of EM components. When the dissimilarity measure is less than the threshold, the components are combined into a single cluster. The setting can be used only for EM Clustering. A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out. Default is 2.

Table 42-14 (Cont.) Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE EMCS_LINKAGE_AVERAGE EMCS_LINKAGE_COMPLETE	Allows the specification of a linkage function for the agglomerative clustering step. The setting can be used only for EM Clustering. EMCS_LINKAGE_SINGLE uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes. EMCS_LINKAGE_AVERAGE uses the average distance within the branch. There is less chaining effect and the clusters are more compact. EMCS_LINKAGE_COMPLETE uses the maximum distance within the branch. The clusters are smaller and require strong component overlap. Default is EMCS_LINKAGE_SINGLE.

Table 42-15 Expectation Maximization Settings for Cluster Statistics

Setting Name	Setting Value	Description
EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE EMCS_CLUS_STATS_DISABLE	Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced, and GET_MODEL_DETAILS_EM only returns taxonomy (hierarchy) and cluster counts. The setting can be used only for EM Clustering. Default is EMCS_CLUS_STATS_ENABLE.
EMCS_MIN_PCT_ATTR_SUPPORT	TO_CHAR(0 < *numeric_expr* < 1)	Minimum support required for including an attribute in the cluster rule. The support is the percentage of the data rows assigned to a cluster that must have non-null values for the attribute. The setting can be used only for EM Clustering. Default is 0.1.

Table 42-16 Expectation Maximization Settings for Anomaly Detection

Setting Name	Setting Value	Description
EMCS_OUTLIER_RATE	TO_CHAR(0 < numeric_expr < 1)	The desired rate of outliers in the training data. The setting can be used only for EM Anomaly. Default is 0.05.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

42.1.5.5 DBMS_DATA_MINING — Algorithm Settings: Explicit Semantic Analysis

Explicit Semantic Analysis (ESA) is a useful technique for extracting meaningful and interpretable features.

The settings listed in the following table configure the ESA values.

Table 42-17 Explicit Semantic Analysis Settings

Setting Name	Setting Value	Description
ESAS_EMBEDDINGS	ESAS_EMBEDDINGS_ENABLE ESAS_EMBEDDINGS_DISABLE	This setting applies to feature extraction models. The default value is ESAS_EMBEDDINGS_DISABLE. When you set ESAS_EMBEDDINGS_ENABLE: • ESA generates embeddings during scoring • The FEATURE_ID of the generated embeddings is of the datatype NUMBER • The CASE_ID_COLUMN_NAME argument of the DBMS_DATA_MINING.CREATE_MODEL and DBMS_DATA_MINING.CREATE_MODEL2 function is optional. .
ESAS_EMBEDDING_SIZE	A positive integer less than or equal to 4096	This setting applies to feature extraction models. This setting specifies the size of the vectors representing embeddings. You can set this parameter only if you have enabled ESAS_EMBEDDINGS. The default size is 1024. If this value is less than the number of distinct features in the training set, then the actual number of explicit features is used as the size of embedding vectors instead.
ESAS_MIN_ITEMS	Text input 100 Non-text input is 0	This setting determines the minimum number of non-zero entries that need to be present in an input row. The default is 100 for text input and 0 for non-text input.
ESAS_TOPN_FEATURES	A positive integer	This setting controls the maximum number of features per attribute. The default is 1000.
ESAS_VALUE_THRESHOLD	Non-negative number	This setting thresholds a small value for attribute weights in the transformed build data. The default is 1e-8.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.

- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about ESA.

42.1.5.6 DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing

These settings configure the behavior of the Exponential Smoothing (ESM) algorithm.

The settings listed in the following table specify the setting names and possible values for Exponential Smoothing. You can specify the Setting Value using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.EXSM_SIMPLE`. Alternatively, you can specify the Setting Value without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'EXSM_SIMPLE'`.

For Global settings, see DBMS_DATA_MINING — Global Settings.

Table 42-18 Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_MODEL	EXSM_SIMPLE EXSM_SIMPLE_MULT_ERR EXSM_HOLT EXSM_HOLT_DAMPED EXSM_MULT_TREND EXSM_MULT_TREND_DAMPED EXSM_SEASON_ADD EXSM_SEASON_MUL EXSM_WINTERS EXSM_WINTERS_DAMPED EXSM_ADDWINTERS EXSM_ADDWINTERS_DAMPED EXSM_WINTERS_MUL_TREND EXSM_WINTERS_MUL_TREND_DMP	This setting specifies the model. EXSM_SIMPLE: Forecasts data as a weighted moving average, with the influence of past observations declining exponentially with the length of time since the observation occurred. Errors in estimation are assumed to be normally distributed, with constant mean and variance. It is appropriate for data with no clear trend or seasonal pattern. EXSM_SIMPLE_MULT_ERR: Forecasts data as a weighted moving average, with the influence of past observations declining exponentially with the length of time since the observation occurred. Errors in estimation are assumed to be proportional to the level of the prior estimate. EXSM_HOLT: Applies Holt's linear exponential smoothing method, designed to forecast data with an underlying linear trend. EXSM_HOLT_DAMPED: Applies Holt's linear exponential smoothing with a damping factor to progressively reduce the strength of the trend over time. EXSM_MULT_TREND: Applies an exponential smoothing framework with a multiplicative trend component, effectively capturing data where trends are not linear but grow or decay over time. EXSM_MULT_TREND_DAMPED: Applies an exponential smoothing algorithm with a multiplicative trend that diminishes over time, providing a conservative approach to trend estimation. EXSM_SEASON_ADD: Applies an exponential smoothing with an additive seasonal component, isolating and accounting for seasonal variations without incorporating a trend. EXSM_SEASON_MUL: Executes exponential smoothing with a multiplicative seasonal component, capturing seasonal effects that increase or decrease in proportion to the level of the series. EXSM_WINTERS: Applies the Holt-Winters method with additive trends and multiplicative seasonality, offering a robust model for data with both linear trend and proportional seasonal variation. EXSM_WINTERS_DAMPED: Applies the Holt-Winters method with a damped trend and multiplicative seasonality, moderating the

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
		linear trend over time while still capturing proportional seasonal changes. `EXSM_ADDWINTERS`: Applies the Holt-Winters additive model to simultaneously smooth data with linear trends and additive seasonal effects. `EXSM_ADDWINTERS_DAMPED`: Applies the Holt-Winters additive approach with a damping mechanism, reducing the impact of the trend and seasonal components over time. `EXSM_WINTERS_MULT_TREND`: Applies the Holt-Winters model with both trend and seasonality components being multiplicative, suited for series where the seasonal variations and trends are both increasing or decreasing proportional to level. `EXSM_WINTERS_MUL_TREND_DMP`: Applies the Holt-Winters model with a damped multiplicative trend, effectively moderating the exponential increase or decrease of both trend and seasonal components over time. The default value is `EXSM_SIMPLE`.
`EXSM_SEASONALITY`	`positive integer > 1`	This setting specifies a positive integer value as the length of seasonal cycle. The value it takes must be larger than `1`. For example, setting value `4` means that every group of four observations forms a seasonal cycle. This setting is only applicable and must be provided for models with seasonality, otherwise the model throws an error. When `EXSM_INTERVAL` is not set, this setting applies to the original input time series. When `EXSM_INTERVAL` is set, this setting applies to the accumulated time series.

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_INTERVAL	EXSM_INTERVAL_YEAR EXSM_INTERVAL_QTR EXSM_INTERVAL_MONTH EXSM_INTERVAL_WEEK EXSM_INTERVAL_DAY EXSM_INTERVAL_HOUR EXSM_INTERVAL_MINUTE EXSM_INTERVAL_SECOND	This setting only applies and must be provided when the time column (case_id column) has datetime type. It specifies the spacing interval of the accumulated equally spaced time series. The model throws an error if the time column of input table is of datetime type and setting EXSM_INTERVAL is not provided. The model throws an error if the time column of input table is of oracle number type and setting EXSM_INTERVAL is provided. EXSM_INTERVAL_YEAR: This option sets the spacing interval of the accumulated time series to one year. When selected, the data is aggregated or summarized on a yearly basis. EXSM_INTERVAL_QTR: This option sets the spacing interval to a quarter, aggregating the data for every three months. EXSM_INTERVAL_MONTH: This option adjusts the spacing interval to one month. The accumulated time series represent aggregated or summarized data for each month. EXSM_INTERVAL_WEEK: With this option data is aggregated or summarized on a weekly basis, setting the spacing interval to one week. EXSM_INTERVAL_DAY: This option adjusts the spacing interval to one day. It's suitable for scenarios where daily aggregated insights are required. EXSM_INTERVAL_HOUR: For more granular insights, this option sets the spacing interval to one hour. It's especially useful when analyzing data that changes significantly within a day. EXSM_INTERVAL_MINUTE: With this option the spacing is set to one minute. This provides a very detailed view of data, suitable for applications like high-frequency trading or real-time monitoring systems. EXSM_INTERVAL_SECOND: For most granular details, this options sets the spacing interval to one second. It's tailored for scenarios requiring real-time or near-real-time analysis.

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
`EXSM_INITVL_OPTIMIZE`	`EXSM_INITVL_OPTIMIZE_ENABLE` `EXSM_INITVL_OPTIMIZE_DISABLE`	The setting `EXSM_INITVL_OPTIMIZE` determines whether initial values are optimized during model build. The default value is `EXSM_INITVL_OPTIMIZE_ENABLE`. **Note:** `EXSM_INITVL_OPTIMIZE` can only be set to `EXSM_INITVL_OPTIMIZE_DISABLE` if the user has set `EXSM_MODEL` to `EXSM_HW` or `EXSM_HW_ADDSEA`. If `EXSM_MODEL` is set to another model type or is not specified, error 40213 (conflicting settings) is thrown and the model is not built.

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_ACCUMULATE	EXSM_ACCU_TOTAL EXSM_ACCU_STD EXSM_ACCU_MAX EXSM_ACCU_MIN EXSM_ACCU_AVG EXSM_ACCU_MEDIAN EXSM_ACCU_COUNT	This setting only applies and must be provided when the time column has datetime type. It specifies how to generate the value of the accumulated time series from the input time series. EXSM_ACCU_TOTAL: This option calculates the total sum of the time series values within a specified interval. When selected, it will aggregate the data by summing up all the individual values in the datetime range. EXSM_ACCU_STD: This option computes the standard deviation of the time series values within a specified interval. It helps you understand the amount of variation or dispersion in your data. EXSM_ACCU_MAX: By selecting this option, the maximum value of the time series within a specified interval will be determined. It helps in identifying the peak value in the given range. EXSM_ACCU_MIN: This option focuses on determining the minimum value of the time series within a specified interval. It is useful for identifying the lowest value in the time series for the given datetime range. EXSM_ACCU_AVG: This specifies the average value of your time series within a specified interval. It calculates the mean value of all data points in the specified range. EXSM_ACCU_MEDIAN: This option provides the median of the time series values within the given interval. The median gives a central value, which can be especially useful if your data contains outliers. EXSM_ACCU_COUNT: This option counts the number of time series values within the specified interval. It is helpful if you want to know how many data points are present in a certain datetime range. The default value is EXSM_ACCU_TOTAL.

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_SETMISSING	Specify an option: EXSM_MISS_MIN EXSM_MISS_MAX EXSM_MISS_AVG EXSM_MISS_MEDIAN EXSM_MISS_LAST EXSM_MISS_FIRST EXSM_MISS_PREV EXSM_MISS_NEXT EXSM_MISS_AUTO	This setting specifies how to handle missing values, which may come from input data and/or the accumulation process of time series. You can specify either a number or an option. If a number is specified, all the missing values are set to that number. EXSM_MISS_MIN: Replaces missing value with minimum of the accumulated time series. EXSM_MISS_MAX: Replaces missing value with maximum of the accumulated time series. EXSM_MISS_AVG: Replaces missing value with average of the accumulated time series. EXSM_MISS_MEDIAN: Replaces missing value with median of the accumulated time series. EXSM_MISS_LAST: Replaces missing value with last non-missing value of the accumulated time series. EXSM_MISS_FIRST: Replaces missing value with first non-missing value of the accumulated time series. EXSM_MISS_PREV: Replaces missing value with the previous non-missing value of the accumulated time series. EXSM_MISS_NEXT: Replaces missing value with the next non-missing value of the accumulated time series. EXSM_MISS_AUTO: EXSM model treats the input data as an irregular (non-uniformly spaced) time series. If this setting is not provided, EXSM_MISS_AUTO is the default value. In such a case, the model treats the input time series as irregular time series, viewing missing values as gaps.
EXSM_PREDICTION_STEP	It must be set to a number between 1-30.	This setting specifies how many steps ahead the predictions are to be made. If it is not set, the default value is 1: the model gives one-step-ahead prediction. A value greater than 30 results in an error.
EXSM_CONFIDENCE_LEVEL	It must be a number between 0 and 1, exclusive.	This setting specifies the desired confidence level for prediction. The lower and upper bounds of the specified confidence interval is reported. If this setting is not specified, the default confidence level is 95%.

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description

`EXSM_OPT_CRITERION`	`EXSM_OPT_CRIT_LIK`
	`EXSM_OPT_CRIT_MSE`
	`EXSM_OPT_CRIT_AMSE`
	`EXSM_OPT_CRIT_SIG`
	`EXSM_OPT_CRIT_MAE`
		This setting specifies the desired optimization criterion. The optimization criterion is useful as a diagnostic for comparing models' fit to the same data.
		`EXSM_OPT_CRIT_LIK`: This represents the negative double of the logarithm of the likelihood associated with a given model.
		`EXSM_OPT_CRIT_MSE`: This provides the mean squared error pertaining to the model.
		`EXSM_OPT_CRIT_AMSE`: This denotes the average of the mean squared error over a time window as specified by the user.
		`EXSM_OPT_CRIT_SIG`: This metric captures the standard deviation of the residuals of the model.
		`EXSM_OPT_CRIT_MAE`: This metric conveys the average absolute error associated with the model. It measures the size of the error.
		The default value is `EXSM_OPT_CRIT_LIK`.

`EXSM_NMSE`	positive integer	This setting specifies the length of the window used in computing the error metric average mean square error (AMSE).

Table 42-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
`EXSM_SERIES_LIST`	Comma delimited list of time series columns	This setting allows you to forecast up to twenty predictor series in addition to the target series. The column names in `EXSM_SERIES_LIST` are enclosed in single quotes. **Note:** The list is enclosed in single quotes, not the individual column names. For example: `INSERT INTO <settings_table_name VALUES(dbms_data_mining.exsm_series_list, '<column1>,<column2>,<column3>,<column4>');` The prefix `DM$` must be added to the build and scoring data sets. The column names must be less than 125 characters long. See Model Detail Views for Exponential Smoothing.
`EXSM_BACKCAST_OUTPUT`	`EXSM_BACKCAST_OUTPUT_ENABLE` `EXSM_BACKCAST_OUTPUT_DISABLE`	This setting enables the user to optionally suppress the output of backcast values. Backcasts are the model estimates for historical data. See Backcasts in Time Series for information on backcasts. Suppressing the output of backcast values can provide a potentially large reduction in the memory and storage requirements for a partitioned ESM model with a huge number of partitions. The default value is `EXSM_BACKCAST_OUTPUT_ENABLE`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.

- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about ESM.
>
> https://github.com/oracle-samples/oracle-db-examples/tree/main/machine-learning/sql browse to the release folder and click the `oml4sql-time-series-exponential-smoothing.sql` example.

42.1.5.7 DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model

The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.

Table 42-19 DBMS_DATA_MINING GLM Settings

Setting Name	Setting Value	Description
`GLMS_CONF_LEVEL`	`TO_CHAR(0< numeric_expr <1)`	The confidence level for coefficient confidence intervals. The default confidence level is `0.95`.
`GLMS_FTR_GEN_METHOD`	`GLMS_FTR_GEN_QUADRATIC` `GLMS_FTR_GEN_CUBIC`	Whether feature generation is quadratic or cubic. When feature generation is enabled, the algorithm automatically chooses the most appropriate feature generation method based on the data.
`GLMS_FTR_GENERATION`	`GLMS_FTR_GENERATION_ENABLE` `GLMS_FTR_GENERATION_DISABLE`	Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled. **Note:** Feature generation can only be enabled when feature selection is also enabled.

`GLMS_FTR_SEL_CRIT`	`GLMS_FTR_SEL_AIC`
	`GLMS_FTR_SEL_SBIC`
	`GLMS_FTR_SEL_RIC`
	`GLMS_FTR_SEL_ALPHA_INV`
		Feature selection penalty criterion for adding a feature to the model.
		When feature selection is enabled, the algorithm automatically chooses the penalty criterion based on the data.

`GLMS_FTR_SELECTION`	`GLMS_FTR_SELECTION_ENABLE` `GLMS_FTR_SELECTION_DISABLE`	Whether or not feature selection is enabled for GLM. By default, feature selection is not enabled.
`GLMS_MAX_FEATURES`	`TO_CHAR(0 < numeric_expr <= 2000)`	When feature selection is enabled, this setting specifies the maximum number of features that can be selected for the final model. By default, the algorithm limits the number of features to ensure sufficient memory.

Table 42-19 (Cont.) DBMS_DATA_MINING GLM Settings

Setting Name	Setting Value	Description
GLMS_PRUNE_MODEL	GLMS_PRUNE_MODEL_ENABLE GLMS_PRUNE_MODEL_DISABLE	Prune enable or disable for features in the final model. Pruning is based on T-Test statistics for linear regression, or Wald Test statistics for logistic regression. Features are pruned in a loop until all features are statistically significant with respect to the full data. When feature selection is enabled, the algorithm automatically prunes based on the data.
GLMS_REFERENCE_CLASS_NAME	*target_value*	The target value used as the reference class in a binary logistic regression model. Probabilities are produced for the other class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.
GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	Enable or disable ridge regression. Ridge applies to both regression and classification machine learning functions. When ridge is enabled, prediction bounds are not produced by the PREDICTION_BOUNDS SQL function. **Note**: Ridge may only be enabled when feature selection is not specified, or has been explicitly disabled. If ridge regression and feature selection are both explicitly enabled, then an exception is raised.
GLMS_RIDGE_VALUE	TO_CHAR (*numeric_expr* > 0)	The value of the ridge parameter. This setting is only used when the algorithm is configured to use ridge regression. If ridge regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm.
GLMS_ROW_DIAGNOSTICS	GLMS_ROW_DIAG_ENABLE GLMS_ROW_DIAG_DISABLE (default).	Enable or disable row diagnostics.
GLMS_CONV_TOLERANCE	The range is (0, 1) non-inclusive.	Convergence Tolerance setting of the GLM algorithm The default value is system-determined.
GLMS_NUM_ITERATIONS	Positive integer	Maximum number of iterations for the GLM algorithm. The default value is system-determined.
GLMS_BATCH_ROWS	0 or Positive integer	Number of rows in a batch used by the SGD solver. The value of this parameter sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 2000

Table 42-19 (Cont.) DBMS_DATA_MINING GLM Settings

Setting Name	Setting Value	Description
GLMS_SOLVER	GLMS_SOLVER_SGD (StochasticGradient Descent) GLMS_SOLVER_CHOL (Cholesky) GLMS_SOLVER_QR GLMS_SOLVER_LBFGS_ADMM	This setting allows the user to choose the GLM solver. The solver cannot be selected if GLMS_FTR_SELECTION setting is enabled. The following are the options: • GLMS_SOLVER_SGD: Optimizes generalized linear models by iteratively updating parameters using a subset of the data to minimize errors. : • GLMS_SOLVER_CHOL: Solves generalized linear models using the Cholesky decomposition method, which provides a stable and efficient solution by transforming the model into a simpler form. • GLMS_SOLVER_QR: Utilizes the QR decomposition technique to solve generalized linear models, ensuring numerical stability and accuracy by decomposing the problem into orthogonal and triangular components. • GLMS_SOLVER_LBFGS_ADMM: Combines L-BFGS, an approximation of the Broyden-Fletcher-Goldfarb-Shanno optimization algorithm, with ADMM for solving large-scale generalized linear model problems efficiently. The default value is system determined.
GLMS_SPARSE_SOLVER	GLMS_SPARSE_SOLVER_ENABLE GLMS_SPARSE_SOLVER_DISABLE (default).	This setting allows the user to use sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.
GLMS_LINK_FUNCTION	GLMS_IDENTITY_LINK GLMS_LOGIT_LINK GLMS_PROBIT_LINK GLMS_CLOGLOG_LINK GLMS_CAUCHIT_LINK	This setting allows the user to specify the link function for building a GLM model. The link functions are specific to the mining function. For classification, the following are applicable: • GLMS_LOGIT_LINK (default) • GLMS_PROBIT_LINK • GLMS_CLOGLOG_LINK • GLMS_CAUCHIT_LINK For regression, the following is applicable: • GLMS_IDENTITY_LINK (default)

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.
- DBMS_DATA_MINING — Solver Settings: ADMM
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.
- *Oracle Machine Learning for SQL Concepts*

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about GLM.

42.1.5.8 DBMS_DATA_MINING — Algorithm Settings: *k*-Means

The settings listed in the following table configure the behavior of the *k*-Means algorithm.

Table 42-20 k-Means Settings

Setting Name	Setting Value	Description
KMNS_CONV_TOLERANCE	TO_CHAR(0<*numeric_expr*<1)	Minimum Convergence Tolerance for *k*-Means. The algorithm iterates until the minimum Convergence Tolerance is satisfied or until the maximum number of iterations, specified in KMNS_ITERATIONS, is reached. Decreasing the Convergence Tolerance produces a more accurate solution but may result in longer run times. The default Convergence Tolerance is 0.001.
KMNS_DISTANCE	KMNS_COSINE KMNS_EUCLIDEAN	Distance function for *k*-Means. The default distance function is KMNS_EUCLIDEAN.
KMNS_ITERATIONS	TO_CHAR(*positive_numeric_expr*)	Maximum number of iterations for *k*-Means. The algorithm iterates until either the maximum number of iterations is reached or the minimum Convergence Tolerance, specified in KMNS_CONV_TOLERANCE, is satisfied. The default number of iterations is 20.
KMNS_MIN_PCT_ATTR_SUPPORT	TO_CHAR(0<=*numeric_expr*<=1)	Minimum percentage of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster. If the data is sparse or includes many missing values, a minimum support that is too high can cause very short rules or even empty rules. The default minimum support is 0.1.

Table 42-20 (Cont.) k-Means Settings

Setting Name	Setting Value	Description
KMNS_NUM_BINS	TO_CHAR(*numeric_expr*>0)	Number of bins in the attribute histogram produced by *k*-means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value that have only one bin. The default number of histogram bins is 11.
KMNS_SPLIT_CRITERION	KMNS_SIZE KMNS_VARIANCE	Split criterion for *k*-means. The split criterion controls the initialization of new *k*-Means clusters. The algorithm builds a binary tree and adds one new cluster at a time. When the split criterion is based on size, the new cluster is placed in the area where the largest current cluster is located. When the split criterion is based on the variance, the new cluster is placed in the area of the most spread-out cluster. The default split criterion is the KMNS_VARIANCE.
KMNS_RANDOM_SEED	Non-negative integer	This setting controls the seed of the random generator used during the *k*-Means initialization. It must be a non-negative integer value. The default is 0.
KMNS_DETAILS	KMNS_DETAILS_NONE KMNS_DETAILS_HIERARCHY KMNS_DETAILS_ALL	This setting determines the level of cluster detail that are computed during the build. KMNS_DETAILS_NONE: No cluster details are computed. Only the scoring information is persisted. KMNS_DETAILS_HIERARCHY: Cluster hierarchy and cluster record counts are computed. This is the default value. KMNS_DETAILS_ALL: Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed.

Table 42-20 (Cont.) k-Means Settings

Setting Name	Setting Value	Description
KMNS_WINSORIZE	KMNS_WINSORIZE_ENABLE KMNS_WINSORIZE_DISABLE	To winorize data, enable or disable this parameter. Data is restricted in a window size of six standard deviations around the mean value when winsorize is enabled. This functionality can be used with AUTO_DATA_PREP turned ON and OFF. The values outside the range are replaced with the ends of the interval. Winsorize is not enabled by default. **Note:** Winsorize is only available when the KMNS_EUCLIDEAN distance function is used. An exception is raised if Winsorize is enabled and other distance functions are set.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

- For generic machine learning function settings related to Clustering, see DBMS_DATA_MINING — Machine Learning Functions.
- *Oracle Machine Learning for SQL Concepts* for information about *k*-Means

42.1.5.9 DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test

Settings that configure the training calibration behavior of the Multivariate State Estimation Technique - Sequential Probability Ratio Test algorithm.

Table 42-21 MSET-SPRT Settings

Setting Name	Setting Value	Description
MSET_ADB_HEIGHT	A positive double	Estimates the band within which signal values normally oscillate. The default value is 0.05.
MSET_ALERT_COUNT	A positive integer	The number of the last *n* signals (the alert window) that should have passed the threshold to raise an alert. The alert count should be lower or equal to the alert window. The default value is 5.
MSET_ALERT_WINDOW	A positive integer greater than or equal to MSET_ALERT_COUNT	The number of signals to consider in the SPRT hypothesis consolidation logic. The default value is 5.
MSET_ALPHA_PROB	A positive double between 0 and 1	False Alarm Probability FAP (false positive). The default is 0.01.
MSET_BETA_PROB	A positive double between 0 and 1	Missed Alarm Probability MAP (false negative). The default is 0.10.
MSET_HELDASIDE	A positive integer	The approximate number of data rows used for MSET model calibration. You can use ODMS_RANDOM_SEED to change the held-aside sample. The default value is 10000.
MSET_MEMORY_VECTORS	A positive integer	The default value is data driven.
MSET_PROJECTION_THRESHOLD	A positive integer >0, <=10000	Specifies whether to use random projections. When the number of sensors exceeds the setting value, random projections are used. To turn off random projections, set the threshold to a value that is equal to or greater than the number of sensors. The default value is 500.
MSET_STD_TOLERANCE	A positive integer	The tolerance in standard deviations used in the SPRT calculation. The default value is 3.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

42.1.5.10 DBMS_DATA_MINING — Algorithm Settings: Naive Bayes

The settings listed in the following table configure the behavior of the Naive Bayes algorithm.

Table 42-22 Naive Bayes Settings

Setting Name	Setting Value	Description
`NABS_PAIRWISE_THRESHOLD`	`TO_CHAR(0<= numeric_expr <=1)`	Value of pairwise threshold for NB algorithm Default is `0`.
`NABS_SINGLETON_THRESHOLD`	`TO_CHAR(0<= numeric_expr <=1)`	Value of singleton threshold for NB algorithm Default value is `0`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about Naive Bayes

42.1.5.11 DBMS_DATA_MINING — Algorithm Settings: Neural Network

The settings listed in the following table configure the behavior of the Neural Network algorithm.

Table 42-23 DBMS_DATA_MINING Neural Network Settings

Setting Name	Setting Value	Description
`NNET_SOLVER`	One of the following strings: • `NNET_SOLVER_ADAM` • `NNET_SOLVER_LBFGS`	Specifies the method of optimization. The default value is system determined.

Table 42-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Setting Value	Description

NNET_ACTIVATIONS	One or more of the following strings:
	• NNET_ACTIVATIONS_ARCTAN
	• NNET_ACTIVATIONS_BIPOLAR_SIG
	• NNET_ACTIVATIONS_LINEAR
	• NNET_ACTIVATIONS_LOG_SIG
	• NNET_ACTIVATIONS_RELU
	• NNET_ACTIVATIONS_TANH
		Specifies the activation functions for the hidden layers. You can specify a single activation function, which is then applied to each hidden layer, or you can specify an activation function for each layer individually. Different layers can have different activation functions.
		To apply a different activation function to one or more of the layers, you must specify an activation function for each layer. The number of activation functions you specify must be consistent with the NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER values.
		For example, if you have three hidden layers, you could specify the use of the same activation function for all three layers with the following settings value:
		`('NNET_ACTIVATIONS', 'NNET_ACTIVATIONS_TANH')`
		The following settings value specifies a different activation function for each layer:
		`('NNET_ACTIVATIONS', '''NNET_ACTIVATIONS_TANH'', ''NNET_ACTIVATIONS_LOG_SIG'', ''NNET_ACTIVATIONS_ARCTAN''')`
		Note: You specify the different activation functions as strings within a single string. All quotes are single and two single quotes are used to escape a single quote in SQL statements and PL/SQL blocks.
		The default value is NNET_ACTIVATIONS_LOG_SIG.

NNET_HELDASIDE_MAX_FAIL	A positive integer	With NNET_REGULARIZER_HELDASIDE, the training process is stopped early if the network performance on the validation data fails to improve or remains the same for NNET_HELDASIDE_MAX_FAIL epochs in a row. The default value is 6.
NNET_HELDASIDE_RATIO	0 <= *numeric_expr* <=1	Define the held ratio for the held-aside method. The default value is 0.25.

Table 42-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Setting Value	Description
NNET_HIDDEN_LAYERS	A positive integer	Defines the topology by the number of hidden layers. The default value is 1.
NNET_ITERATIONS	A positive integer	Specifies the maximum number of iterations in the Neural Network algorithm. For the DMSSET_NN_SOLVER_LBFGS solver, the default value is 200. For the DMSSET_NN_SOLVER_ADAM solver, the default value is 10000.
NNET_NODES_PER_LAYER	A positive integer or a list of positive integers	Defines the topology by the number of nodes per layer. Different layers can have different numbers of nodes. To specify the same number of nodes for each layer, you can provide a single value, which is then applied to each layer. To specify a different number of nodes for one or more layers, provide a list of comma-separated positive integers, one for each layer. For example, '10, 20, 5' for three layers. The setting values must be consistent with the NNET_HIDDEN_LAYERS value. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes > 50).
NNET_REG_LAMBDA	TO_CHAR(*numeric_expr* >=0)	Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1.
NNET_REGULARIZER	One of the following strings: • NNET_REGULARIZER_HELDASIDE • NNET_REGULARIZER_L2 • NNET_REGULARIZER_NONE	Regularization setting for Neural Network algorithm. If the total number of training rows is greater than 50000, the default is NNET_REGULARIZER_HELDASIDE. If the total number of training rows is less than or equal to 50000, the default is NNET_REGULARIZER_NONE.
NNET_TOLERANCE	TO_CHAR(0< *numeric_expr* <1)	Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001.

Table 42-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Setting Value	Description
NNET_WEIGHT_LOWER_BOUND	A real number	The setting specifies the lower bound of the region where weights are randomly initialized. NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND must be set together. Setting one and not setting the other raises an error. NNET_WEIGHT_LOWER_BOUND must not be greater than NNET_WEIGHT_UPPER_BOUND. The default value is –sqrt(6/(l_nodes+r_nodes)). The value of l_nodes for: • input layer dense attributes is (1+number of dense attributes) • input layer sparse attributes is number of sparse attributes • each hidden layer is (1+number of nodes in that hidden layer) The value of r_nodes is the number of nodes in the layer that the weight is connecting to.
NNET_WEIGHT_UPPER_BOUND	A real number	This setting specifies the upper bound of the region where weights are initialized. It should be set in pairs with NNET_WEIGHT_LOWER_BOUND and its value must not be smaller than the value of NNET_WEIGHT_LOWER_BOUND. If not specified, the values of NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND are system determined. The default value is sqrt(6/(l_nodes+r_nodes)). See NNET_WEIGHT_LOWER_BOUND.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

See Also:

Oracle Machine Learning for SQL Concepts for information about Neural Network.

42.1.5.12 DBMS_DATA_MINING — Algorithm Settings: Non-Negative Matrix Factorization

The settings listed in the following table configure the behavior of the Non-negative Matrix Factorization algorithm.

You can query the data dictionary view *_MINING_MODEL_SETTINGS (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Database Reference* for information about *_MINING_MODEL_SETTINGS.

Table 42-24 NMF Settings

Setting Name	Setting Value	Description
NMFS_CONV_TOLERANCE	TO_CHAR(0< *numeric_expr* <=0.5)	Convergence tolerance for NMF algorithm Default is 0.05
NMFS_NONNEGATIVE_SCORING	NMFS_NONNEG_SCORING_ENABLE NMFS_NONNEG_SCORING_DISABLE	Whether negative numbers should be allowed in scoring results. When set to NMFS_NONNEG_SCORING_ENABLE, negative feature values will be replaced with zeros. When set to NMFS_NONNEG_SCORING_DISABLE, negative feature values will be allowed. Default is NMFS_NONNEG_SCORING_ENABLE
NMFS_NUM_ITERATIONS	TO_CHAR(1 <= *numeric_expr* <=500)	Number of iterations for NMF algorithm Default is 50
NMFS_RANDOM_SEED	TO_CHAR(*numeric_expr*)	Random seed for NMF algorithm. Default is -1.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about NMF

42.1.5.13 DBMS_DATA_MINING — Algorithm Settings: O-Cluster

The settings in the table configure the behavior of the O-Cluster algorithm.

Table 42-25 O-CLuster Settings

Setting Name	Setting Value	Description
`OCLT_SENSITIVITY`	`TO_CHAR(0 <=`*`numeric_expr`*`<=1)`	A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Default is `0.5`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about O-Cluster

42.1.5.14 DBMS_DATA_MINING — Algorithm Settings: Random Forest

These settings configure the behavior of the Random Forest algorithm. Random Forest makes use of the Decision Tree settings to configure the construction of individual trees.

Table 42-26 Random Forest Settings

Setting Name	Setting Value	Description
`RFOR_MTRY`	`a number >= 0`	Size of the random subset of columns to be considered when choosing a split at a node. For each node, the size of the pool remains the same, but the specific candidate columns change. The default is half of the columns in the model signature. The special value `0` indicates that the candidate pool includes all columns.
`RFOR_NUM_TREES`	`1<=` *`a number`* `<=65535`	Number of trees in the forest Default is `20`.
`RFOR_SAMPLING_RATIO`	`0<` *`a fraction`*`<=1`	Fraction of the training data to be randomly sampled for use in the construction of an individual tree. The default is half of the number of rows in the training data.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
- DBMS_DATA_MINING — Algorithm Settings: Decision Tree
These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual decision tree.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about Random Forest

42.1.5.15 DBMS_DATA_MINING — Algorithm Constants and Settings: Singular Value Decomposition

The following settings configure the behavior of the Singular Value Decomposition algorithm.

Table 42-27 Singular Value Decomposition Settings

Setting Name	Setting Value	Description
`SVDS_U_MATRIX_OUTPUT`	`SVDS_U_MATRIX_ENABLE` `SVDS_U_MATRIX_DISABLE`	Indicates whether or not to persist the **U** Matrix produced by SVD. The U matrix in SVD has as many rows as the number of rows in the build data. To avoid creating a large model, the U matrix is persisted only when `SVDS_U_MATRIX_OUTPUT` is enabled. When `SVDS_U_MATRIX_OUTPUT` is enabled, the build data must include a case ID. If no case ID is present and the U matrix is requested, then an exception is raised. Default is `SVDS_U_MATRIX_DISABLE`.
`SVDS_SCORING_MODE`	`SVDS_SCORING_SVD` `SVDS_SCORING_PCA`	Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections will be the same as the U matrix. When the build data is scored with PCA, the projections will be the product of the U and S matrices. Default is `SVDS_SCORING_SVD`.

Table 42-27 (Cont.) Singular Value Decomposition Settings

Setting Name	Setting Value	Description
SVDS_SOLVER	SVDS_SOLVER_TSSVD SVDS_SOLVER_TSEIGEN SVDS_SOLVER_SSVD SVDS_SOLVER_STEIGEN	This setting indicates the solver to be used for computing SVD of the data. In the case of PCA, the solver setting indicates the type of SVD solver used to compute the PCA for the data. When this setting is not specified the solver type selection is data driven. If the number of attributes is greater than 3240, then the default wide solver is used. Otherwise, the default narrow solver is selected. The following are the group of solvers: • Narrow data solvers: for matrices with up to 11500 attributes (TSEIGEN) or up to 8100 attributes (TSSVD). • Wide data solvers: for matrices up to 1 million attributes. For narrow data solvers: • Tall-Skinny SVD uses QR computation TSVD (SVDS_SOLVER_TSSVD) • Tall-Skinny SVD uses eigenvalue computation, TSEIGEN (SVDS_SOLVER_TSEIGEN), is the default solver for narrow data. For wide data solvers: • Stochastic SVD uses QR computation SSVD (SVDS_SOLVER_SSVD), is the default solver for wide data solvers. • Stochastic SVD uses eigenvalue computations, STEIGEN (SVDS_SOLVER_STEIGEN).
SVDS_TOLERANCE	Range [0, 1]	This setting is used to prune features. Define the minimum value the eigenvalue of a feature as a share of the first eigenvalue to not to prune. Default value is data driven.
SVDS_RANDOM_SEED	Range [0 - 4,294,967,296]	The random seed value is used for initializing the sampling matrix used by the Stochastic SVD solver. The default is 0. The SVD Solver must be set to SSVD or STEIGEN.
SVDS_OVER_SAMPLING	Range [1, 5000].	This setting is configures the number of columns in the sampling matrix used by the Stochastic SVD solver. The number of columns in this matrix is equal to the requested number of features plus the oversampling setting. The SVD Solver must be set to SSVD or STEIGEN.
SVDS_POWER_ITERATIONS	Range [0, 20].	The power iteration setting improves the accuracy of the SSVD solver. The default is 2. The SVD Solver must be set to SSVD or STEIGEN.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts

42.1.5.16 DBMS_DATA_MINING — Algorithm Settings: Support Vector Machine

The settings listed in the following table configure the behavior of the Support Vector Machine algorithm.

Table 42-28 SVM Settings

Setting Name	Setting Value	Description
SVMS_COMPLEXITY_FACTOR	TO_CHAR(*numeric_expr* >0)	Regularization setting that balances the complexity of the model against model robustness to achieve good generalization on new data. SVM uses a data-driven approach to finding the complexity factor. Value of complexity factor for SVM algorithm (both classification and regression). Default value estimated from the data by the algorithm.
SVMS_CONV_TOLERANCE	TO_CHAR(*numeric_expr* >0)	Convergence tolerance for SVM algorithm. Default is 0.0001.
SVMS_EPSILON	TO_CHAR(*numeric_expr* >0)	Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data. Value of epsilon factor for SVM regression. Default is 0.1.
SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN SVMS_LINEAR	Kernel for Support Vector Machine. Linear or Gaussian. The default value is SVMS_LINEAR.
SVMS_OUTLIER_RATE	TO_CHAR(*0< numeric_expr* <1)	The desired rate of outliers in the training data. Valid for One-Class SVM models only (anomaly detection). Default is 0.01.
SVMS_STD_DEV	TO_CHAR(*numeric_expr* >0)	Controls the spread of the Gaussian kernel function. SVM uses a data-driven approach to find a standard deviation value that is on the same scale as distances between typical cases. Value of standard deviation for SVM algorithm. This is applicable only for Gaussian kernel. Default value estimated from the data by the algorithm.
SVMS_NUM_ITERATIONS	Positive integer	This setting sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.
SVMS_NUM_PIVOTS	Range [1; 10000]	This setting sets an upper limit on the number of pivots used in the Incomplete Cholesky decomposition. It can be set only for non-linear kernels. The default value is 200.
SVMS_BATCH_ROWS	Positive integer	This setting applies to SVM models with linear kernel. This setting sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 20000.

Table 42-28 (Cont.) SVM Settings

Setting Name	Setting Value	Description
`SVMS_REGULARIZER`	`SVMS_REGULARIZER_L1` `SVMS_REGULARIZER_L2`	This setting controls the type of regularization that the SGD SVM solver uses. The setting can be used only for linear SVM models. The default is system determined because it depends on the potential model size.
`SVMS_SOLVER`	`SVMS_SOLVER_SGD` (Sub-Gradient Descend) `SVMS_SOLVER_IPM` (Interior Point Method)	This setting allows the user to choose the SVM solver. The SGD solver cannot be selected if the kernel is non-linear. The default value is system determined.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about SVM

42.1.5.17 DBMS_DATA_MINING — Algorithm Settings: XGBoost

Settings that configure the behavior of the XGBoost gradient boosting algorithm.

The XGBoost settings are case sensitive. Enter the settings as they appear in the settings table. These settings match the XGBoost settings available in open source. OML4SQL XGBoost is based on the 1.7.4 version of XGBoost.
For Global settings, see DBMS_DATA_MINING — Global Settings.

For generic machine learning technique settings, see DBMS_DATA_MINING — Machine Learning Functions.

Table 42-29 General Settings

Setting Name	Setting Value	Description

`booster`	A string that is one of the following:
	`dart`
	`gblinear`
	`gbtree`
		The booster to use:
		• `dart`
		• `gblinear`
		• `gbtree`
		The `dart` and `gbtree` boosters use tree-based models whereas `gblinear` uses linear functions.
		The default value is `gbtree`.

`num_round`	A non-negative integer.	The number of rounds for boosting. The default value is `10`.

Table 42-30 Settings for Tree Boosting

Setting Name	Setting Value	Description
`alpha`	A non-negative number	L1 regularization term on weights. Increasing this value makes the model more conservative. The default value is `0`.
`colsample_bylevel`	A number in the range (0, 1]	Subsample ratio of columns for each split, in each level. Subsampling occurs each time a new split is made. This parameter has no effect when `tree_method` is set to `hist`. The default value is `1`.
`colsample_bynode`	A number in the range (0, 1]	The subsample ratio of columns for each node (split). Subsampling occurs once every time a new split is evaluated. Columns are subsampled from the set of columns chosen for the current level. The default value is `1`.
`colsample_bytree`	A number in the range (0, 1]	Subsample ratio of columns when constructing each tree. Subsampling occurs once in every boosting iteration. The default value is `1`.
`eta`	A number in the range [0, 1]	Step-size shrinkage used in the update step to prevent overfitting. After each boosting step, `eta` shrinks the feature weights to make the boosting process more conservative. The default value is `0.3`.
`gamma`	A number in the range [0, ∞]	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma value is, the more conservative the algorithm is. The default value is `0`.

Table 42-30 (Cont.) Settings for Tree Boosting

Setting Name	Setting Value	Description
`grow_policy`	A string; one of the following: • `depthwise` • `lossguide`	Controls the way new nodes are added to the tree: • `depthwise` splits at nodes closest to the root • `lossguide` splits at nodes with the highest loss change Valid only if `tree_method` is set to `hist`. The default value is `depthwise`.
`xgboost_interaction_constraints`	`[[x0,x1,x2],[x0,x4],[x5,x6]]` where `xn` are feature names or columns	This setting specifies permitted interactions in the model. Specify the constrains in the form of a nested list where each inner list is a group of features (column names) that are allowed to interact with each other. If a single column is passed in the interactions then, the input is ignored. Here, features x0, x1, and x2 are allowed to interact with each other but with no other feature. Similarly, x0 and x4 are allowed to interact with each other but with no other feature and so on. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.
`lambda`	A non-negative number	L2 regularization term on weights. The default value is `1`.
`max_bin`	A non-negative integer	Maximum number of discrete bins to bucket continuous features. Increasing this number improves the optimality of splits at the cost of higher computation time. This parameter is valid only when `tree_method` is set to `hist`. The default value is `256`.
`max_delta_step`	A number in the range [0, ∞]	Maximum delta step allowed for each leaf output. Setting this to a positive value can help make the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when the class is extremely imbalanced. Setting it to value from 1 to 10 might help control the update. The default value is `0`, which means there is no constraint.

Table 42-30 (Cont.) Settings for Tree Boosting

Setting Name	Setting Value	Description
`max_depth`	An integer in the range [0, ∞]	Maximum depth of a tree. Increasing this value makes the model more complex and more likely to overfit. Setting this to 0 indicates no limit. **Note:** You must set a `max_depth` limit when the `grow_policy` setting is `depthwise`. The default value is `6`.
`max_leaves`	A non-negative number	Maximum number of nodes to add. Use this setting only when `grow_policy` is set to `lossguide`. The default value is `0`.
`min_child_weight`	A number in the range [0, ∞]	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with a sum of instance weight less than `min_child_weight`, then the building process stops partitioning. In a linear regression task, this corresponds to the minimum number of instances needed in each node. The larger `min_child_weight` is, the more conservative the algorithm is. The default value is `1`.
`xgboost_decrease_constraints`	`[x4,x5]`	This setting specifies the features (column names) that must obey decreasing constraint. The feature names are separated by a comma. For example, setting value 'x4,x5' sets decreasing constraint on features x4 and x5. This setting applies to numeric columns and 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.

Table 42-30 (Cont.) Settings for Tree Boosting

Setting Name	Setting Value	Description
`xgboost_increase_constraints`	[`x0,x3`]	This setting specifies the features (column names) that must obey increasing constraint. The feature names are separated by a comma. For example, setting value 'x0,x3' sets increasing constraint on features x0 and x3. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.
`num_parallel_tree`	A non-negative integer	Number of parallel trees constructed during each iteration. Use this option to support a boosted random forest. The default value is `1`.
`scale_pos_weight`	A non-negative number	Controls the balance of positive and negative weights, which is useful for unbalanced classes. A typical value to consider: `sum(negative cases) / sum(positive cases)`. The default value is `1`.
`sketch_eps`	A number in the range (0, 1)	Increases enumeration accuracy. Valid only for the approximate greedy tree method. Compared to directly selecting the number of bins, this setting comes with a theoretical guarantee with sketch accuracy. You usually do not need to change this setting, but you might consider setting a lower number for more accurate enumeration. The default value is `0.03`.
`subsample`	A number in the range (0, 1]	Subsample ratio of the training instances. A setting of 0.5 means that XGBoost randomly samples half of the training data prior to growing trees, which prevents overfitting. Subsampling occurs once in every boosting iteration. The default value is `1`.

Table 42-30 (Cont.) Settings for Tree Boosting

Setting Name	Setting Value	Description
`tree_method`	A string that is one of the following: • `approx` • `auto` • `exact` • `hist`	Tree construction algorithm used in XGBoost: • `approx`: Approximate greedy algorithm using sketching and histogram. • `auto`: Use a heuristic to choose the faster algorithm: – For a small to medium sized data set, uses the exact greedy algorithm. – For a very large data set, uses the approximate greedy algorithm. • `exact`: Exact greedy algorithm. • `hist`: Fast histogram optimized approximate greedy algorithm; uses some performance improvements such as bins caching. The default value is `auto`.
`updater`	A comma-separated string; one or more of the following: • `grow_colmaker` • `grow_histmaker` • `grow_skmaker` • `grow_quantile_histmaker` • `prune` • `sync`	Defines the sequence of tree updaters to run, which provides a modular way to construct and to modify the trees. This is an advanced parameter that is usually set automatically, depending on some other parameters. However, you can also explicitly specify a settting. The setting values are: • `grow_colmaker`: Non-distributed column-based construction of trees. • `grow_histmaker`: Distributed tree construction with row-based data splitting based on a global proposal of histogram counting. • `grow_skmaker`: Uses the approximate sketching algorithm. • `grow_quantile_histmaker`: Grow tree using quantized histogram. • `prune`: Prunes the splits where loss < `min_split_loss` (or `gamma`). • `sync`: Synchronizes trees in all distributed nodes.

Table 42-31 Settings for the Dart Booster

Setting Name	Setting Value	Description
`one_drop`	A number that is 0 or 1	When set to 1, at least one tree is always dropped during the dropout. When set to 0, at least one tree is not always dropped during the dropout. The default value is `0`.

Table 42-31 (Cont.) Settings for the Dart Booster

Setting Name	Setting Value	Description
`normalize_type`	A string; either: • `forest` • `tree`	Type of normalization algorithm: • `forest`: New trees have the same weight as the sum of the dropped trees (forest): – The weight of new trees is `1 / (1 + learning_rate)` – Dropped trees are scaled by a factor of `1 / (1 + learning_rate)` • `tree`: New trees have the same weight as dropped trees: – The weight of new trees is `1 / (k + learning_rate)` – Dropped trees are scaled by a factor of `k / (k + learning_rate)` The default value is `tree`.
`rate_drop`	A number in the range [0.0, 1.0]	Dropout rate (a fraction of the previous trees to drop during the dropout). The default value is `0.0`.
`sample_type`	A string; either: • `uniform` • `weighted`	Type of sampling algorithm: • `uniform`: Dropped trees are selected uniformly • `weighted`: Dropped trees are selected in proportion to weight The default value is `uniform`.
`skip_drop`	A number in the range [0.0, 1.0]	Probability of skipping the dropout procedure during a boosting iteration. If a dropout is skipped, new trees are added in the same manner as `gbtree`. A non-zero `skip_drop` has higher priority than `rate_drop` or `one_drop`. The default value is `0.0`.

Table 42-32 Settings for the Linear Booster

Setting Name	Setting Value	Description
`alpha`	A non-negative number	L1 regularization term on weights, normalized to the number of training examples. Increasing this value makes the model more conservative. The default value is `0`.

Table 42-32 (Cont.) Settings for the Linear Booster

Setting Name	Setting Value	Description
`feature_selector`	A string that is one of the following: • `cyclic` • `greedy` • `random` • `shuffle` • `thrifty`	Feature selection and ordering method: • `cyclic`: Deterministic selection by cycling through the features one at a time. • `greedy`: Selects the coordinate with the greatest gradient magnitude. This method: – Has `O(num_feature^2)` complexity – Is fully deterministic – Allows restricting the selection to the `top_k` features per group with the largest magnitude of univariate weight change, by setting the `top_k` parameter; doing so reduces the complexity to `O(num_feature*top_k)`. • `random`: A random (with replacement) coordinate selector. • `shuffle`: Similar to `cyclic` but with random feature shuffling prior to each update. • `thrifty`: Thrifty, approximately-greedy feature selector. Prior to cyclic updates, reorders features in descending magnitude of their univariate weight changes. This operation is multithreaded and is a linear complexity approximation of the quadratic greedy selection. Restricts the selection per group to the `top_k` features with the largest magnitude of univariate weight change. The default value is `cyclic`.
`lambda`	A non-negative number	L2 regularization term on weights, normalized to the number of training examples. Increasing this value makes the model more conservative. The default value is `0`.
`top_k`	A non-negative integer	Number of top features to select for the `greedy` or `thrifty` feature selector. The value of 0 uses all of the features. The default value is `0`.
`updater`	A string that is one of the following: • `coord_descent` • `shotgun`	Algorithm to fit the linear model: • `coord_descent`: Ordinary coordinate descent algorithm; multithreaded but still produces a deterministic solution. • `shotgun`: Parallel coordinate descent algorithm based on the `shotgun` algorithm; uses "hogwild" parallelism and therefore produces a nondeterministic solution on each run. The default value is `shotgun`.

Table 42-33 Settings for Tweedie Regression

Setting Name	Setting Value	Description
`tweedie_variance_power`	A number in the range (1, 2)	Controls the variance of the Tweedie distribution `var(y) ~ E(y)^tweedie_variance_power`. A setting closer to 1 shifts towards a Poisson distribution. A setting closer to 2 shifts towards a gamma distribution. The default value is `1.5`.

Some XGBoost objectives apply only to classification function models and other objectives apply only to regression function models. If you specify an incompatible `objective` value, an error is raised. In the `DBMS_DATA_MINING.CREATE_MODEL` procedure, if you specify `DBMS_DATA_MINING.CLASSIFICATION` as the function, then the only objective values that you can use are the `binary` and `multi` values. The one exception is `binary: logitraw`, which produces a continuous value and applies only to a regression model. If you specify `DBMS_DATA_MINING.REGRESSION` as the function, then you can specify `binary: logitraw` or any of the `count`, `rank`, `reg`, and `survival` values as the objective.

Table 42-34 Settings for Learning Tasks

Setting Name	Setting Value	Description
`objective`	For a classification model, a string that is one of the following: • `binary:hinge` • `binary:logistic` • `multi:softmax` • `multi:softprob` For a regression model, a string that is one of the following: • `binary:logitraw` • `count:poisson` • `rank:map` • `rank:ndcg` • `rank:pairwise` • `reg:gamma` • `reg:logistic` • `reg:tweedie` • `survival:aft` • `survival:cox` • `reg:squarederror` • `reg:squaredlogerror`	**Settings for a Classification model:** • `binary:hinge`: Hinge loss for binary classification. This setting makes predictions of 0 or 1, rather than producing probabilities. • `binary:logistic`: Logistic regression for binary classification. The output is the probability. • `multi:softmax`: Performs multiclass classification using the `softmax` objective; you must also set `num_class(`*`number_of_classes`*`)`. • `multi:softprob`: : Same as `softmax`, except the output is a vector of `ndata * nclass`, which can be further reshaped to an `ndata * nclass` matrix. The result contains the predicted probability of each data point belonging to each class. The default `objective` value for classification is `multi:softprob`. **Settings for a Regression model:** • `binary:logitraw`: Logistic regression for binary classification; the output is the score before logistic transformation. • `count:poisson`: Poisson regression for count data; the output is the mean of the Poisson distribution. The `max_delta_step` value is set to 0.7 by default in Poisson regression to safeguard optimization. • `rank:map`: Using `LambdaMART`, performs list-wise ranking in which the Mean Average Precision (MAP) is maximized. • `rank:ndcg`: Using `LambdaMART`, performs list-wise ranking in which the Normalized Discounted Cumulative Gain (NDCG) is maximized. • `rank:pairwise`: Performs ranking by minimizing the pairwise loss. • `reg:gamma`: Gamma regression with log-link; the output is the mean of the gamma distribution. This setting might be useful for any outcome that might be gamma-distributed, such as modeling insurance claims severity. • `reg:logistic`: Logistic regression. • `reg:tweedie`: Tweedie regression with log-link. This setting might be useful for any outcome that might be Tweedie-distributed, such as modeling total loss in insurance. • `survival:aft`: Applies the Accelerated Failure Time (AFT) model for censored survival time data. When you select this option, `eval_metric` uses `aft-nloglik` as the default value. • `survival:cox`: Cox regression for right-censored survival time data (negative values are

Table 42-34 (Cont.) Settings for Learning Tasks

Setting Name	Setting Value	Description
		considered right-censored). Predictions are returned on the hazard ratio scale (that is, as `HR = exp(marginal_prediction)` in the proportional hazard function `h(t) = h0(t) * HR)`. • `reg:squarederror`: Regression with squared loss. • `reg:squaredlogerror`: Regression with squared log loss. All input labels must be greater than -1. The default `objective` value for regression is `reg:squarederror`.
`xgboost_aft_loss_distribution`	[normal, logistic, extreme]	Specifies the distribution of the Z term in the AFT model. It specifies the Probabilty Density Function used by `survival:aft` objective and `aft-nloglik` evaluation metric. The default value is `normal`.
`xgboost_aft_loss_distribution_scale`	A positive number	Specifies the scaling factor σ, which scales the size of Z term in the AFT model. The default value is `1`.
`xgboost_aft_right_bound_column_name`	*column_name*	Specifies the column containing the right bounds of the labels for an AFT model. You cannot select this parameter for a non-AFT model. **Note:** Oracle Machine Learning does not support `BOOLEAN` values for this setting.
`base_score`	A number	Initial prediction score of all instances, global bias. For a sufficient number of iterations, changing this value does not have much effect. The default value is `0.5`.

Table 42-34 (Cont.) Settings for Learning Tasks

Setting Name	Setting Value	Description
`eval_metric`	A comma-separated string; one or more of the following: • `aft-nloglik` • `auc` • `aucpr` • `cox-nloglik` • `error` • `error@`*`t`* • `gamma-deviance` • `gamma-nloglik` • `logloss` • `mae` • `map` • `map@`*`n`* • `merror` • `mlogloss` • `ndcg` • `ndcg@`*`n`* • `poisson-nloglik` • `rmse` • `tweedie-nloglik@`*`rho`* • `ndcg-` • `map-` • `rmsle`	Evaluation metrics for validation data. You can specify one or more of these evaluation metrics: • `aft-nloglik`: Sets the `eval_metric` to negative log likelihood of AFT model. • `auc`: Area under the curve. • `aucpr`: Area under the PR curve. • `cox-nloglik`: Negative partial log-likelihood for Cox proportional hazards regression. • `error`: Binary classification error rate, calculated as the number of wrong cases divided by the number of all cases. For the predictions, the evaluation regards the instances with a prediction value larger than 0.5 as positive instances, and the others as negative instances. • `error@`*`t`*: You can specify a binary classification threshold value other than 0.5 by specifying a numerical value *t*; for example, `error@0.8`. • `gamma-deviance`: Residual deviance for `gamma` regression. • `gamma-nloglik`: Negative log-likelihood for `gamma` regression. • `logloss`: Negative log-likelihood. • `mae`: Mean absolute error. • `map`: Mean average precision. • `map@`*`n`*: Assigns the integer *n* as the cut-off value for the top positions in the lists for evaluation. • `merror`: Multiclass classification error rate calculated as the number of wrong cases divided by the number of all cases; the objective must be `multi:softprob` or `multi:softmax`. • `mlogloss`: Multiclass `logloss`; the objective must be `multi:softprob` or `multi:softmax`. • `ndcg`: Normalized Discounted Cumulative Gain. • `ndcg@`*`n`*: Assigns the integer *n* as the cut-off value for the top positions in the lists for evaluation. • `poisson-nloglik`: Negative log-likelihood for Poisson regression • `rmse`: Root Mean Square Error. • `tweedie-nloglik@`*`rho`*: Negative log-likelihood for Tweedie regression (at a specified value `rho` of the `tweedie_variance_power` parameter); `rho` must be a number in the range (1, 2); for example, `tweedie-nloglik@1.8`. • `ndcg-` and `map-`: In XGBoost, NDCG and MAP will evaluate the score of a list without any positive samples as 1. By adding "-" in the evaluation metric XGBoost will evaluate these score as 0 to be consistent under some conditions.

Table 42-34 (Cont.) Settings for Learning Tasks

Setting Name	Setting Value	Description
		• `rmsle`: It is root mean square log error. This is the default metric of `reg:squaredlogerror` objective. This metric reduces errors generated by outliers in dataset. But because log function is employed, `rmsle` might output nan when prediction value is less than -1. A default metric is assigned according to the objective: • `error` for classification • `mean average precision` for ranking • `rmse` for regression
`seed`	A non-negative integer	Random number seed. The default value is `0`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql/, select the release, and browse for an example of XGBoost.

42.1.6 DBMS_DATA_MINING — Solver Settings

Oracle Machine Learning for SQL algorithms can use different solvers. Solver settings can be provided at build time in the settings table.

Related Topics

- DBMS_DATA_MINING - Solver Settings: Adam
  These settings configure the behavior of the Adaptive Moment Estimation (Adam) solver.
- DBMS_DATA_MINING — Solver Settings: ADMM
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

42.1.6.1 DBMS_DATA_MINING - Solver Settings: Adam

These settings configure the behavior of the Adaptive Moment Estimation (Adam) solver.

Neural Network models use these settings.

Table 42-35 DBMS_DATA_MINING Adam Settings

Setting Name	Setting Value	Description
`ADAM_ALPHA`	A non-negative double precision floating point number in the interval (0; 1]	The learning rate for Adam. The default value is `0.001`.
`ADAM_BATCH_ROWS`	A positive integer	The number of rows per batch. The default value is `10000`.
`ADAM_BETA1`	A positive double precision floating point number in the interval [0; 1)	The exponential decay rate for the 1st moment estimates. The default value is `0.9`.
`ADAM_BETA2`	A positive double precision floating point number in the interval [0; 1)	The exponential decay rate for the 2nd moment estimates. The default value is `0.99`.
`ADAM_GRADIENT_TOLERANCE`	A positive double precision floating point number	The gradient infinity norm tolerance for Adam. The default value is `1E-9`.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.

42.1.6.2 DBMS_DATA_MINING — Solver Settings: ADMM

The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.

Table 42-36 DBMS_DATA_MINING ADMM Settings

Settings Name	Setting Value	Description
`ADMM_CONSENSUS`	A positive integer	It is a ADMM's consensus parameter. The value must be a positive number. The default value is `0.1`.
`ADMM_ITERATIONS`	A positive integer	The number of `ADMM` iterations. The value must be a positive integer. The default value is `50`.
`ADMM_TOLERANCE`	A positive integer	It is a tolerance parameter. The value must be a positive number. The default value is `0.0001`

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model
  The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.
- *Oracle Machine Learning for SQL Concepts*

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about neural network

42.1.6.3 DBMS_DATA_MINING — Solver Settings: LBFGS

The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

Table 42-37 DBMS_DATA_MINING L-BFGS Settings

Setting Name	Setting Value	Description
LBFGS_GRADIENT_TOLERANCE	TO_CHAR (*numeric_expr* >0)	Defines gradient infinity norm tolerance for L-BFGS. Default value is 1E-9.
LBFGS_HISTORY_DEPTH	The value must be a positive integer.	Defines the number of historical copies kept in L-BFGS solver. The default value is 20.
LBFGS_SCALE_HESSIAN	LBFGS_SCALE_HESSIAN_ENABLE LBFGS_SCALE_HESSIAN_DISABLE	Defines whether to scale Hessian in L-BFGS or not. Default value is LBFGS_SCALE_HESSIAN_ENABLE.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.
- DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model
  The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about neural network

42.1.7 DBMS_DATA_MINING Datatypes

The `DBMS_DATA_MINING` package defines object data types for processing transactional data. The package also defines a type for user-specified transformations. These types are called `DM_NESTED_n`, where *n* identifies the Oracle data type of the nested attributes.

The Oracle Machine Learning for SQL object data types are described in the following table:

Table 42-38 DBMS_DATA_MINING Summary of Data Types

Datatype	Description
`DM_NESTED_BINARY_DOUBLE`	The name and value of a numerical attribute of type `BINARY_DOUBLE`.
`DM_NESTED_BINARY_DOUBLES`	A collection of `DM_NESTED_BINARY_DOUBLE`.
`DM_NESTED_BINARY_FLOAT`	The name and value of a numerical attribute of type `BINARY_FLOAT`.
`DM_NESTED_BINARY_FLOATS`	A collection of `DM_NESTED_BINARY_FLOAT`.
`DM_NESTED_CATEGORICAL`	The name and value of a categorical attribute of type `CHAR`, `VARCHAR`, or `VARCHAR2`.
`DM_NESTED_CATEGORICALS`	A collection of `DM_NESTED_CATEGORICAL`.
`DM_NESTED_NUMERICAL`	The name and value of a numerical attribute of type `NUMBER` or `FLOAT`.
`DM_NESTED_NUMERICALS`	A collection of `DM_NESTED_NUMERICAL`.
`ORA_MINING_VARCHAR2_NT`	A table of `VARCHAR2(4000)`.
`TRANSFORM_LIST`	A list of user-specified transformations for a model. Accepted as a parameter by the CREATE_MODEL Procedure. This collection type is defined in the DBMS_DATA_MINING_TRANSFORM package.

For more information about processing nested data, see *Oracle Machine Learning for SQL User's Guide*.

> **Note:**
>
> Starting from Oracle Database 12*c* Release 2, `*GET_MODEL_DETAILS` are deprecated and are replaced with *Model Detail Views*. See *Oracle Machine Learning for SQL User's Guide*.

42.1.7.1 Deprecated Types

This topic contains tables listing deprecated types.

The `DBMS_DATA_MINING` package defines object datatypes for storing information about model attributes. Most of these types are returned by the table functions `GET_n`, where *n* identifies the type of information to return. These functions take a model name as input and return the requested information as a collection of rows.

For a list of the `GET` functions, see "Summary of DBMS_DATA_MINING Subprograms".

All the table functions use pipelining, which causes each row of output to be materialized as it is read from model storage, without waiting for the generation of the complete table object. For more information on pipelined, parallel table functions, consult the *Oracle Database PL/SQL Language Reference.*

Table 42-39 DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
DM_CENTROID	The centroid of a cluster.
DM_CENTROIDS	A collection of DM_CENTROID. A member of DM_CLUSTER.
DM_CHILD	A child node of a cluster.
DM_CHILDREN	A collection of DM_CHILD. A member of DM_CLUSTER.
DM_CLUSTER	A cluster. A cluster includes DM_PREDICATES, DM_CHILDREN, DM_CENTROIDS, and DM_HISTOGRAMS. It also includes a DM_RULE. See also, DM_CLUSTER Fields.
DM_CLUSTERS	A collection of DM_CLUSTER. Returned by GET_MODEL_DETAILS_KM Function, GET_MODEL_DETAILS_OC Function, and GET_MODEL_DETAILS_EM Function. See also, DM_CLUSTER Fields.
DM_CONDITIONAL	The conditional probability of an attribute in a Naive Bayes model.
DM_CONDITIONALS	A collection of DM_CONDITIONAL. Returned by GET_MODEL_DETAILS_NB Function.
DM_COST_ELEMENT	The actual and predicted values in a cost matrix.
DM_COST_MATRIX	A collection of DM_COST_ELEMENT. Returned by GET_MODEL_COST_MATRIX Function.
DM_EM_COMPONENT	A component of an Expectation Maximization model.
DM_EM_COMPONENT_SET	A collection of DM_EM_COMPONENT. Returned by GET_MODEL_DETAILS_EM_COMP Function.
DM_EM_PROJECTION	A projection of an Expectation Maximization model.
DM_EM_PROJECTION_SET	A collection of DM_EM_PROJECTION. Returned by GET_MODEL_DETAILS_EM_PROJ Function.
DM_GLM_COEFF	The coefficient and associated statistics of an attribute in a Generalized Linear Model.
DM_GLM_COEFF_SET	A collection of DM_GLM_COEFF. Returned by GET_MODEL_DETAILS_GLM Function.
DM_HISTOGRAM_BIN	A histogram associated with a cluster.
DM_HISTOGRAMS	A collection of DM_HISTOGRAM_BIN. A member of DM_CLUSTER. See also, DM_CLUSTER Fields.
DM_ITEM	An item in an association rule.
DM_ITEMS	A collection of DM_ITEM.
DM_ITEMSET	A collection of DM_ITEMS.

Table 42-39 (Cont.) DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
DM_ITEMSETS	A collection of DM_ITEMSET. Returned by GET_FREQUENT_ITEMSETS Function.
DM_MODEL_GLOBAL_DETAIL	High-level statistics about a model.
DM_MODEL_GLOBAL_DETAILS	A collection of DM_MODEL_GLOBAL_DETAIL. Returned by GET_MODEL_DETAILS_GLOBAL Function.
DM_NB_DETAIL	Information about an attribute in a Naive Bayes model.
DM_NB_DETAILS	A collection of DM_DB_DETAIL. Returned by GET_MODEL_DETAILS_NB Function.
DM_NMF_ATTRIBUTE	An attribute in a feature of a Non-Negative Matrix Factorization model.
DM_NMF_ATTRIBUTE_SET	A collection of DM_NMF_ATTRIBUTE. A member of DM_NMF_FEATURE.
DM_NMF_FEATURE	A feature in a Non-Negative Matrix Factorization model.
DM_NMF_FEATURE_SET	A collection of DM_NMF_FEATURE. Returned by GET_MODEL_DETAILS_NMF Function.
DM_PREDICATE	Antecedent and consequent in a rule.
DM_PREDICATES	A collection of DM_PREDICATE. A member of DM_RULE and DM_CLUSTER. Predicates are returned by GET_ASSOCIATION_RULES Function, GET_MODEL_DETAILS_EM Function, GET_MODEL_DETAILS_KM Function, and GET_MODEL_DETAILS_OC Function. See also, DM_CLUSTER Fields.
DM_RANKED_ATTRIBUTE	An attribute ranked by its importance in an Attribute Importance model.
DM_RANKED_ATTRIBUTES	A collection of DM_RANKED_ATTRIBUTE. Returned by GET_MODEL_DETAILS_AI Function.
DM_RULE	A rule that defines a conditional relationship. The rule can be one of the association rules returned by GET_ASSOCIATION_RULES Function, or it can be a rule associated with a cluster in the collection of clusters returned by GET_MODEL_DETAILS_KM Function and GET_MODEL_DETAILS_OC Function. See also, DM_CLUSTER Fields.
DM_RULES	A collection of DM_RULE. Returned by GET_ASSOCIATION_RULES Function. See also, DM_CLUSTER Fields.
DM_SVD_MATRIX	A factorized matrix S, V, or U returned by a Singular Value Decomposition model.
DM_SVD_MATRIX_SET	A collection of DM_SVD_MATRIX. Returned by GET_MODEL_DETAILS_SVD Function.
DM_SVM_ATTRIBUTE	The name, value, and coefficient of an attribute in a Support Vector Machine model.

Table 42-39 (Cont.) DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
`DM_SVM_ATTRIBUTE_SET`	A collection of `DM_SVM_ATTRIBUTE`. Returned by GET_MODEL_DETAILS_SVM Function. Also a member of `DM_SVM_LINEAR_COEFF`.
`DM_SVM_LINEAR_COEFF`	The linear coefficient of each attribute in a Support Vector Machine model.
`DM_SVM_LINEAR_COEFF_SET`	A collection of `DM_SVM_LINEAR_COEFF`. Returned by GET_MODEL_DETAILS_SVM Function for an SVM model built using the linear kernel.
`DM_TRANSFORM`	The transformation and reverse transformation expressions for an attribute.
`DM_TRANSFORMS`	A collection of `DM_TRANSFORM`. Returned by GET_MODEL_TRANSFORMATIONS Function.

Return Values for Clustering Algorithms

The table contains description of `DM_CLUSTER` return value columns, nested table columns, and rows.

Table 42-40 DM_CLUSTER Return Values for Clustering Algorithms

Return Value	Description

`DM_CLUSTERS`	A set of rows of type `DM_CLUSTER`. The rows have the following columns:
	`(id NUMBER,`
	`cluster_id VARCHAR2(4000),`
	`record_count NUMBER,`
	`parent NUMBER,`
	`tree_level NUMBER,`
	`dispersion NUMBER,`
	`split_predicate DM_PREDICATES,`
	`child DM_CHILDREN,`
	`centroid DM_CENTROIDS,`
	`histogram DM_HISTOGRAMS,`
	`rule DM_RULE)`

`DM_PREDICATE`	The `antecedent` and `consequent` columns each return nested tables of type `DM_PREDICATES`. The rows, of type `DM_PREDICATE`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`

DM_CLUSTER Fields

The following table describes `DM_CLUSTER` fields.

Table 42-41 DM_CLUSTER Fields

Column Name	Description
id	Cluster identifier
cluster_id	The ID of a cluster in the model
record_count	Specifies the number of records
parent	Parent ID
tree_level	Specifies the number of splits from the root
dispersion	A measure used to quantify whether a set of observed occurrences are dispersed compared to a standard statistical model.

split_predicate	The split_predicate column of DM_CLUSTER returns a nested table of type DM_PREDICATES. Each row, of type DM_PREDICATE, has the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`conditional_operator CHAR(2) /`
	`*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`
	
	Note: The Expectation Maximization algorithm uses all the fields except dispersion and split_predicate.

child	The child column of DM_CLUSTER returns a nested table of type DM_CHILDREN. The rows, of type DM_CHILD, have a single column of type NUMBER, which contains the identifiers of each child.

centroid	The centroid column of DM_CLUSTER returns a nested table of type DM_CENTROIDS. The rows, of type DM_CENTROID, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`mean NUMBER,`
	`mode_value VARCHAR2(4000),`
	`variance NUMBER)`

histogram	The histogram column of DM_CLUSTER returns a nested table of type DM_HISTOGRAMS. The rows, of type DM_HISTOGRAM_BIN, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`bin_id NUMBER,`
	`lower_bound NUMBER,`
	`upper_bound NUMBER,`
	`label VARCHAR2(4000),`
	`count NUMBER)`

Table 42-41 (Cont.) DM_CLUSTER Fields

Column Name	Description

`rule`	The `rule` column of `DM_CLUSTER` returns a single row of type `DM_RULE`. The columns are:
	`(rule_id INTEGER,`
	`antecedent DM_PREDICATES,`
	`consequent DM_PREDICATES,`
	`rule_support NUMBER,`
	`rule_confidence NUMBER,`
	`rule_lift NUMBER,`
	`antecedent_support NUMBER,`
	`consequent_support NUMBER,`
	`number_of_items INTEGER)`

Usage Notes

- The table function pipes out rows of type `DM_CLUSTER`. For information on Oracle Machine Learning for SQL data types and piped output from table functions, see "Data Types".
- For descriptions of predicates (`DM_PREDICATE`) and rules (`DM_RULE`), see GET_ASSOCIATION_RULES Function.

42.1.8 Summary of DBMS_DATA_MINING Subprograms

This table summarizes the subprograms included in the `DBMS_DATA_MINING` package.

The `GET_*` interfaces are replaced by model views. Oracle recommends that users leverage model detail views instead. For more information, refer to Model Detail Views in *Oracle Machine Learning for SQL User's Guide* and Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES in *Oracle Database Reference*.

Table 42-42 DBMS_DATA_MINING Package Subprograms

Subprogram	Purpose
ADD_COST_MATRIX Procedure	Adds a cost matrix to a classification model
ADD_PARTITION Procedure	Adds single or multiple partitions in an existing partition model
ALTER_REVERSE_EXPRESSION Procedure	Changes the reverse transformation expression to an expression that you specify
APPLY Procedure	Applies a model to a data set (scores the data)
COMPUTE_CONFUSION_MATRIX Procedure	Computes the confusion matrix for a classification model
COMPUTE_CONFUSION_MATRIX_PART Procedure	Computes the evaluation matrix for partitioned models
COMPUTE_LIFT Procedure	Computes lift for a classification model
COMPUTE_LIFT_PART Procedure	Computers lift for partitioned models
COMPUTE_ROC Procedure	Computes Receiver Operating Characteristic (ROC) for a classification model

Table 42-42 (Cont.) DBMS_DATA_MINING Package Subprograms

Subprogram	Purpose
COMPUTE_ROC_PART Procedure	Computes Receiver Operating Characteristic (ROC) for a partitioned model
CREATE_MODEL Procedure	Creates a model
CREATE_MODEL2 Procedure	Creates a model without extra persistent stages
Create Model Using Registration Information	Fetches setting information from JSON object
DROP_ALGORITHM Procedure	Drops the registered algorithm information.
DROP_PARTITION Procedure	Drops a single partition
DROP_MODEL Procedure	Drops a model
EXPORT_MODEL Procedure	Exports a model to a dump file
EXPORT_SERMODEL Procedure	Exports a model in a serialized format
FETCH_JSON_SCHEMA Procedure	Fetches and reads JSON schema from `all_mining_algorithms` view
GET_MODEL_COST_MATRIX Function	Returns the cost matrix for a model
IMPORT_MODEL Procedure	Imports a model into a user schema
IMPORT_ONNX_MODEL Procedure	Imports an ONNX model into the Database
IMPORT_SERMODEL Procedure	Imports a serialized model back into the database
JSON Schema for R Extensible Algorithm	Displays flexibility in creating JSON schema for R Extensible
REGISTER_ALGORITHM Procedure	Registers a new algorithm
RANK_APPLY Procedure	Ranks the predictions from the `APPLY` results for a classification model
REMOVE_COST_MATRIX Procedure	Removes a cost matrix from a model
RENAME_MODEL Procedure	Renames a model

Deprecated GET_MODEL_DETAILS

Starting from Oracle Database 12*c* Release 2, the following `GET_MODEL_DETAILS` are deprecated:

Table 42-43 Deprecated `GET_MODEL_DETAILS` Functions

Subprogram	Purpose
GET_ASSOCIATION_RULES Function	Returns the rules from an association model
GET_FREQUENT_ITEMSETS Function	Returns the frequent itemsets for an association model
GET_MODEL_DETAILS_AI Function	Returns details about an attribute importance model
GET_MODEL_DETAILS_EM Function	Returns details about an Expectation Maximization model

Table 42-43 (Cont.) Deprecated `GET_MODEL_DETAILS` Functions

Subprogram	Purpose
GET_MODEL_DETAILS_EM_COMP Function	Returns details about the parameters of an Expectation Maximization model
GET_MODEL_DETAILS_EM_PROJ Function	Returns details about the projects of an Expectation Maximization model
GET_MODEL_DETAILS_GLM Function	Returns details about a Generalized Linear Model model
GET_MODEL_DETAILS_GLOBAL Function	Returns high-level statistics about a model
GET_MODEL_DETAILS_KM Function	Returns details about a *k*-Means model
GET_MODEL_DETAILS_NB Function	Returns details about a Naive Bayes model
GET_MODEL_DETAILS_NMF Function	Returns details about a Non-Negative Matrix Factorization model
GET_MODEL_DETAILS_OC Function	Returns details about an O-Cluster model
GET_MODEL_SETTINGS Function	Returns the settings used to build the given model This function is replaced with `USER/ALL/DBA_MINING_MODEL_SETTINGS`
GET_MODEL_SIGNATURE Function	Returns the list of columns from the build input table This function is replaced with `USER/ALL/DBA_MINING_MODEL_ATTRIBUTES`
GET_MODEL_DETAILS_SVD Function	Returns details about a Singular Value Decomposition model
GET_MODEL_DETAILS_SVM Function	Returns details about a Support Vector Machine model with a linear kernel
GET_MODEL_TRANSFORMATIONS Function	Returns the transformations embedded in a model This function is replaced with `USER/ALL/DBA_MINING_MODEL_XFORMS`
GET_MODEL_DETAILS_XML Function	Returns details about a Decision Tree model
GET_TRANSFORM_LIST Procedure	Converts between two different transformation specification formats

Related Topics

- *Oracle Machine Learning for SQL User's Guide*
- *Oracle Database Reference*

42.1.8.1 ADD_COST_MATRIX Procedure

The `ADD_COST_MATRIX` procedure associates a cost matrix table with a classification model. The cost matrix biases the model by assigning costs or benefits to specific model outcomes.

The cost matrix is stored with the model and taken into account when the model is scored.

You can also specify a cost matrix inline when you invoke an Oracle Machine Learning for SQL function for scoring. To view the scoring matrix for a model, query the `DM$VC` prefixed model view. Refer to Model Detail View for Classification Algorithm.

To obtain the default scoring matrix for a model, query the `DM$VC` prefixed model view. To remove the default scoring matrix from a model, use the `REMOVE_COST_MATRIX` procedure. See REMOVE_COST_MATRIX Procedure.

See Also:

- "Biasing a Classification Model" in *Oracle Machine Learning for SQL Concepts* for more information about costs
- *Oracle Database SQL Language Reference* for syntax of inline cost matrix
- Specifying Costs in *Oracle Machine Learning for SQL User's Guide*

Syntax

```
DBMS_DATA_MINING.ADD_COST_MATRIX (
      model_name                IN VARCHAR2,
      cost_matrix_table_name    IN VARCHAR2,
      cost_matrix_schema_name   IN VARCHAR2 DEFAULT NULL);
      partition_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-44 ADD_COST_MATRIX Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is assumed.
`cost_matrix_table_name`	Name of the cost matrix table (described in Table 42-45).
`cost_matrix_schema_name`	Schema of the cost matrix table. If no schema is specified, then the current schema is used.
`partition_name`	Name of the partition in a partitioned model

Usage Notes

1. If the model is not in your schema, then `ADD_COST_MATRIX` requires the `ALTER ANY MINING MODEL` system privilege or the `ALTER` object privilege for the machine learning model.
2. The cost matrix table must have the columns shown in Table 42-45.

Table 42-45 Required Columns in a Cost Matrix Table

Column Name	Data Type
`ACTUAL_TARGET_VALUE`	Valid target data type
`PREDICTED_TARGET_VALUE`	Valid target data type
`COST`	`NUMBER`, `FLOAT`, `BINARY_DOUBLE`, or `BINARY_FLOAT`

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for valid target data types

3. The types of the actual and predicted target values must be the same as the type of the model target. For example, if the target of the model is `BINARY_DOUBLE`, then the actual and predicted values must be `BINARY_DOUBLE`. If the actual and predicted values are `CHAR` or `VARCHAR`, then `ADD_COST_MATRIX` treats them as `VARCHAR2` internally.

   If the types do not match, or if the actual or predicted value is not a valid target value, then the `ADD_COST_MATRIX` procedure raises an error.

   > **Note:**
   >
   > If a reverse transformation is associated with the target, then the actual and predicted values must be consistent with the target after the reverse transformation has been applied.
   >
   > See "Reverse Transformations and Model Transparency" under the "About Transformation Lists" section in DBMS_DATA_MINING_TRANSFORM Operational Notes for more information.

4. Since a benefit can be viewed as a negative cost, you can specify a benefit for a given outcome by providing a negative number in the `costs` column of the cost matrix table.
5. All classification algorithms can use a cost matrix for scoring. The Decision Tree algorithm can also use a cost matrix at build time. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See Table 42-7.

   The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one.
6. Scoring on a partitioned model is partition-specific. Scoring cost matrices can be added to or removed from an individual partition in a partitioned model. If `PARTITION_NAME` is NOT NULL, then the model must be a partitioned model. The `COST_MATRIX` is added to that partition of the partitioned model.

   If the `PARTITION_NAME` is NULL, but the model is a partitioned model, then the `COST_MATRIX` table is added to every partition in the model.

Example

This example creates a cost matrix table called `COSTS_NB` and adds it to a Naive Bayes model called `NB_SH_CLAS_SAMPLE`. The model has a binary target: 1 means that the customer responds to a promotion; 0 means that the customer does not respond. The cost matrix assigns a cost of .25 to misclassifications of customers who do not respond and a cost of .75 to misclassifications of customers who do respond. This means that it is three times more costly to misclassify responders than it is to misclassify non-responders.

```
CREATE TABLE costs_nb (
  actual_target_value           NUMBER,
  predicted_target_value        NUMBER,
  cost                          NUMBER);
INSERT INTO costs_nb values (0, 0, 0);
INSERT INTO costs_nb values (0, 1, .25);
INSERT INTO costs_nb values (1, 0, .75);
INSERT INTO costs_nb values (1, 1, 0);
COMMIT;

EXEC dbms_data_mining.add_cost_matrix('nb_sh_clas_sample', 'costs_nb');

SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
   FROM mining_data_apply_v
   WHERE PREDICTION(nb_sh_clas_sample COST MODEL
      USING cust_marital_status, education, household_size) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;

C        CNT    AVG_AGE
- ---------- ----------
F         72         39
M        555         44
```

42.1.8.2 ADD_PARTITION Procedure

`ADD_PARTITION` procedure supports a single or multiple partition addition to an existing partitioned model.

The `ADD_PARTITION` procedure derives build settings and user-defined expressions from the existing model. The target column must exist in the input data query when adding partitions to a supervised model.

Syntax

```
DBMS_DATA_MINING.ADD_PARTITION (
      model_name                 IN VARCHAR2,
      data_query                 IN CLOB,
      add_options                IN VARCHAR2 DEFAULT ERROR);
```

Parameters

Table 42-46 ADD_PARTITION Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`data_query`	An arbitrary SQL statement that provides data to the model build. The user must have privilege to evaluate this query.

Table 42-46 (Cont.) ADD_PARTITION Procedure Parameters

Parameter	Description
`add_options`	Allows users to control the conditional behavior of ADD for cases where rows in the input dataset conflict with existing partitions in the model. The following are the possible values: • `REPLACE`: Replaces the existing partition for which the conflicting keys are found. • `ERROR`: Terminates the ADD operation without adding any partitions. • `IGNORE`: Eliminates the rows having the conflicting keys. **Note:** For better performance, Oracle recommends using `DROP_PARTITION` followed by the `ADD_PARTITION` instead of using the `REPLACE` option.

42.1.8.3 ALTER_REVERSE_EXPRESSION Procedure

This procedure replaces a reverse transformation expression with an expression that you specify. If the attribute does not have a reverse expression, the procedure creates one from the specified expression.

You can also use this procedure to customize the output of clustering, feature extraction, and anomaly detection models.

Syntax

```
DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION (
         model_name              VARCHAR2,
         expression              CLOB,
         attribute_name          VARCHAR2 DEFAULT NULL,
         attribute_subname       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-47 ALTER_REVERSE_EXPRESSION Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used.
`expression`	An expression to replace the reverse transformation associated with the attribute.
`attribute_name`	Name of the attribute. Specify `NULL` if you wish to apply *`expression`* to a cluster, feature, or One-Class SVM prediction.
`attribute_subname`	Name of the nested attribute if *`attribute_name`* is a nested column, otherwise `NULL`.

Usage Notes

1. For purposes of model transparency, Oracle Machine Learning for SQL provides reverse transformations for transformations that are embedded in a model. Reverse transformations are applied to the attributes returned in model detail views and to the scored target of predictive models.

   > **See Also:**
   >
   > - “About Transformation Lists” under DBMS_DATA_MINING_TRANSFORM Operational Notes
   > - Model Detail Views in *Oracle Machine Learning for SQL User’s Guide*

2. If you alter the reverse transformation for the target of a model that has a cost matrix, you must specify a transformation expression that has the same type as the actual and predicted values in the cost matrix. Also, the reverse transformation that you specify must result in values that are present in the cost matrix.

   > **See Also:**
   >
   > "ADD_COST_MATRIX Procedure" and *Oracle Machine Learning for SQL Concepts* for information about cost matrixes.

3. To prevent reverse transformation of an attribute, you can specify `NULL` for *`expression`*.

4. The reverse transformation expression can contain a reference to a PL/SQL function that returns a valid Oracle data type. For example, you could define a function like the following for a categorical attribute named `blood_pressure` that has values 'Low', 'Medium' and 'High'.

```
CREATE OR REPLACE FUNCTION numx(c char) RETURN NUMBER IS
  BEGIN
    CASE c WHEN ''Low'' THEN RETURN 1;
           WHEN ''Medium'' THEN RETURN 2;
           WHEN ''High'' THEN RETURN 3;
           ELSE RETURN null;
    END CASE;
  END numx;
```

   Then you could invoke `ALTER_REVERSE_EXPRESION` for `blood_pressure` as follows.

```
EXEC dbms_data_mining.alter_reverse_expression(
             '<model_name>', 'NUMX(blood_pressure)', 'blood_pressure');
```

5. You can use `ALTER_REVERSE_EXPRESSION` to label clusters produced by clustering models and features produced by feature extraction.

   You can use `ALTER_REVERSE_EXPRESSION` to replace the zeros and ones returned by anomaly-detection models. By default, anomaly-detection models label anomalous records with 0 and all other records with 1.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about anomaly detection

Examples

1. In this example, the target (`affinity_card`) of the model `CLASS_MODEL` is manipulated internally as `yes` or `no` instead of `1` or `0` but returned as `1`s and `0`s when scored. The `ALTER_REVERSE_EXPRESSION` procedure causes the target values to be returned as `TRUE` or `FALSE`.

```
DECLARE
        v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
          'affinity_card', NULL,
          'decode(affinity_card, 1, ''yes'', ''no'')',
          'decode(affinity_card, ''yes'', 1, 0)');
    dbms_data_mining.CREATE_MODEL(
      model_name             => 'CLASS_MODEL',
      mining_function        => dbms_data_mining.classification,
      data_table_name        => 'mining_data_build',
      case_id_column_name    => 'cust_id',
      target_column_name     => 'affinity_card',
      settings_table_name    => NULL,
      data_schema_name       => 'oml_user',
      settings_schema_name   => NULL,
      xform_list             => v_xlst );
  END;
/
SELECT cust_income_level, occupation,
           PREDICTION(CLASS_MODEL USING *) predict_response
      FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
      ORDER BY cust_income_level;

CUST_INCOME_LEVEL              OCCUPATION            PREDICT_RESPONSE
------------------------------ --------------------- --------------------
A: Below 30,000                Transp.                                  1
E: 90,000 - 109,999            Transp.                                  1
E: 90,000 - 109,999            Sales                                    1
G: 130,000 - 149,999           Handler                                  0
G: 130,000 - 149,999           Crafts                                   0
H: 150,000 - 169,999           Prof.                                    1
J: 190,000 - 249,999           Prof.                                    1
J: 190,000 - 249,999           Sales                                    1

BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION (
     model_name       => 'CLASS_MODEL',
     expression       => 'decode(affinity_card, ''yes'', ''TRUE'',
''FALSE'')',
     attribute_name   => 'affinity_card');
END;
/
column predict_response on
column predict_response format a20
SELECT cust_income_level, occupation,
```

```
         PREDICTION(CLASS_MODEL USING *) predict_response
      FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
      ORDER BY cust_income_level;

CUST_INCOME_LEVEL              OCCUPATION            PREDICT_RESPONSE
------------------------------ --------------------- --------------------
A: Below 30,000                Transp.               TRUE
E: 90,000 - 109,999            Transp.               TRUE
E: 90,000 - 109,999            Sales                 TRUE
G: 130,000 - 149,999           Handler               FALSE
G: 130,000 - 149,999           Crafts                FALSE
H: 150,000 - 169,999           Prof.                 TRUE
J: 190,000 - 249,999           Prof.                 TRUE
J: 190,000 - 249,999           Sales                 TRUE
```

2. This example specifies labels for the clusters that result from the `sh_clus` model. The labels consist of the word "Cluster" and the internal numeric identifier for the cluster.

```
BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION( 'sh_clus', '''Cluster ''||value');
END;
/

SELECT cust_id, cluster_id(sh_clus using *) cluster_id
   FROM sh_aprep_num
       WHERE cust_id < 100011
       ORDER by cust_id;

CUST_ID CLUSTER_ID
------- ------------------------------------------------
 100001 Cluster 18
 100002 Cluster 14
 100003 Cluster 14
 100004 Cluster 18
 100005 Cluster 19
 100006 Cluster 7
 100007 Cluster 18
 100008 Cluster 14
 100009 Cluster 8
 100010 Cluster 8
```

42.1.8.4 APPLY Procedure

The `APPLY` procedure applies a machine learning model to the data of interest, and generates the results in a table. The `APPLY` procedure is also referred to as **scoring**.

For predictive machine learning functions, the `APPLY` procedure generates predictions in a target column. For descriptive machine learning functions such as Clustering, the `APPLY` process assigns each case to a cluster with a probability.

In Oracle Machine Learning for SQL, the `APPLY` procedure is not applicable to Association models and Attribute Importance models.

> **Note:**
>
> Scoring can also be performed directly in SQL using the OML4SQL functions. See
>
> - Oracle Machine Learning for SQL Functions in *Oracle Database SQL Language Reference*
> - Scoring and Deployment in *Oracle Machine Learning for SQL User's Guide*

Syntax

```
DBMS_DATA_MINING.APPLY (
      model_name           IN VARCHAR2,
      data_table_name      IN VARCHAR2,
      case_id_column_name  IN VARCHAR2,
      result_table_name    IN VARCHAR2,
      data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-48 ***APPLY Procedure Parameters***

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`data_table_name`	Name of table or view containing the data to be scored
`case_id_column_name`	Name of the case identifier column
`result_table_name`	Name of the table in which to store apply results
`data_schema_name`	Name of the schema containing the data to be scored

Usage Notes

1. The data provided for `APPLY` must undergo the same preprocessing as the data used to create and test the model. When you use Automatic Data Preparation, the preprocessing required by the algorithm is handled for you by the model: both at build time and apply time. (See "Automatic Data Preparation".)
2. `APPLY` creates a table in the user's schema to hold the results. The columns are algorithm-specific.

   The columns in the results table are listed in Table 42-49 through Table 42-53. The case ID column name in the results table will match the case ID column name provided by you. The type of the incoming case ID column is also preserved in `APPLY` output.

> **Note:**
>
> Make sure that the case ID column does not have the same name as one of the columns that will be created by `APPLY`. For example, when applying a Classification model, the case ID in the scoring data must not be `PREDICTION` or `PROBABILITY` (See Table 42-49).

3. The data type for the `PREDICTION`, `CLUSTER_ID`, and `FEATURE_ID` output columns is influenced by any reverse expression that is embedded in the model by the user. If the user does not provide a reverse expression that alters the scored value type, then the types will conform to the descriptions in the following tables. See "ALTER_REVERSE_EXPRESSION Procedure".
4. If the model is partitioned, the `result_table_name` can contain results from different partitions depending on the data from the input data table. An additional column called `PARTITION_NAME` is added to the result table indicating the partition name that is associated with each row.

   For a non-partitioned model, the behavior does not change.

Classification

The results table for Classification has the columns described in Table 42-49. If the target of the model is categorical, the `PREDICTION` column will have a `VARCHAR2` data type. If the target has a binary type, the `PREDICTION` column will have the binary type of the target.

Table 42-49 APPLY Results Table for Classification

Column Name	Data type
`Case ID column name`	Type of the case ID
`PREDICTION`	Type of the target
`PROBABILITY`	`BINARY_DOUBLE`

Anomaly Detection

The results table for Anomaly Detection has the columns described in Table 42-50.

Table 42-50 APPLY Results Table for Anomaly Detection

Column Name	Data Type
`Case ID column name`	Type of the case ID
`PREDICTION`	`NUMBER`
`PROBABILITY`	`BINARY_DOUBLE`

Regression

The results table for Regression has the columns described in APPLY Procedure.

Table 42-51 APPLY Results Table for Regression

Column Name	Data Type
Case ID column name	Type of the case ID
PREDICTION	Type of the target

Clustering

Clustering is an unsupervised machine learning function, and hence there are no targets. The results of an APPLY procedure contain simply the cluster identifier corresponding to a case, and the associated probability. The results table has the columns described in Table 42-52.

Table 42-52 APPLY Results Table for Clustering

Column Name	Data Type
Case ID column name	Type of the case ID
CLUSTER_ID	NUMBER
PROBABILITY	BINARY_DOUBLE

Feature Extraction

Feature Extraction is also an unsupervised machine learning function, hence there are no targets. The results of an APPLY procedure will contain simply the feature identifier corresponding to a case, and the associated match quality. The results table has the columns described in Table 42-53.

Table 42-53 APPLY Results Table for Feature Extraction

Column Name	Data Type
Case ID column name	Type of the case ID
FEATURE_ID	NUMBER
MATCH_QUALITY	BINARY_DOUBLE

Examples

This example applies the GLM Regression model GLMR_SH_REGR_SAMPLE to the data in the MINING_DATA_APPLY_V view. The APPLY results are output of the table REGRESSION_APPLY_RESULT.

```
SQL> BEGIN
       DBMS_DATA_MINING.APPLY (
       model_name     => 'glmr_sh_regr_sample',
       data_table_name     => 'mining_data_apply_v',
       case_id_column_name => 'cust_id',
       result_table_name   => 'regression_apply_result');
    END;
    /

SQL> SELECT * FROM regression_apply_result WHERE cust_id >  101485;
```

```
   CUST_ID PREDICTION
---------- ----------
    101486 22.8048824
    101487 25.0261101
    101488 48.6146619
    101489   51.82595
    101490 22.6220714
    101491 61.3856816
    101492 24.1400748
    101493  58.034631
    101494 45.7253149
    101495 26.9763318
    101496 48.1433425
    101497 32.0573434
    101498 49.8965531
    101499  56.270656
    101500 21.1153047
```

42.1.8.5 COMPUTE_CONFUSION_MATRIX Procedure

This procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

`COMPUTE_CONFUSION_MATRIX` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about confusion matrixes and other test metrics for classification
>
> "COMPUTE_LIFT Procedure"
>
> "COMPUTE_ROC Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
      accuracy                     OUT NUMBER,
      apply_result_table_name      IN  VARCHAR2,
      target_table_name            IN  VARCHAR2,
      case_id_column_name          IN  VARCHAR2,
      target_column_name           IN  VARCHAR2,
      confusion_matrix_table_name  IN  VARCHAR2,
      score_column_name            IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN  VARCHAR2 DEFAULT 'PROBABILITY',
      cost_matrix_table_name       IN  VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN  VARCHAR2 DEFAULT NULL,
      target_schema_name           IN  VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN  VARCHAR2 DEFAULT NULL,
      score_criterion_type         IN  VARCHAR2 DEFAULT 'PROBABILITY');
```

Parameters

Table 42-54 COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
`accuracy`	Output parameter containing the overall percentage accuracy of the predictions.
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`confusion_matrix_table_name`	Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is '`PROBABILITY`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.

Table 42-54 (Cont.) COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `'COSTS'`, the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table. If null, the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `'PROBABILITY'`. To use costs as the scoring criterion, specify `'COST'`. If `score_criterion_type` is set to '`COST`' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_CONFUSION_MATRIX` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to `COMPUTE_CONFUSION_MATRIX`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_CONFUSION_MATRIX` are in a table or view specified in `apply_result_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name             VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 42-55.

Table 42-55 Columns in a Cost Matrix

Column Name	Data Type
actual_target_value	Type of the target column in the build data
predicted_target_value	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
cost	BINARY_DOUBLE

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for valid target data types
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The confusion matrix created by `COMPUTE_CONFUSION_MATRIX` has the columns described in Table 42-56.

Table 42-56 Columns in a Confusion Matrix

Column Name	Data Type
actual_target_value	Type of the target column in the build data
predicted_target_value	Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.
value	BINARY_DOUBLE

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about confusion matrixes

Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`.

Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
       SELECT cust_id,
              PREDICTION(nb_sh_clas_sample USING *) prediction,
```

```
          PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
   FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```
DECLARE
   v_accuracy    NUMBER;
      BEGIN
        DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
                   accuracy                     => v_accuracy,
                   apply_result_table_name      => 'nb_apply_results',
                   target_table_name            => 'mining_data_test_v',
                   case_id_column_name          => 'cust_id',
                   target_column_name           => 'affinity_card',
                   confusion_matrix_table_name  => 'nb_confusion_matrix',
                   score_column_name            => 'PREDICTION',
                   score_criterion_column_name  => 'PROBABILITY'
                   cost_matrix_table_name       =>  null,
                   apply_result_schema_name     =>  null,
                   target_schema_name           =>  null,
                   cost_matrix_schema_name      =>  null,
                   score_criterion_type         => 'PROBABILITY');
        DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
      END;
      /
```

The confusion matrix and model accuracy are shown as follows.

```
 **** MODEL ACCURACY ****: .7847

SQL>SELECT * from nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         60
                  0                      0        891
                  1                      1        286
                  0                      1        263
```

Compute a Confusion Matrix Based on a Cost Matrix Table

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```
SQL> SELECT * from nb_cost_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1        .75
                  1                      0        .25
                  1                      1          0
```

This statement shows how to generate the predictions using `APPLY`.

```
BEGIN
    DBMS_DATA_MINING.APPLY(
          model_name          => 'nb_sh_clas_sample',
          data_table_name     => 'mining_data_test_v',
          case_id_column_name => 'cust_id',
```

```
          result_table_name    => 'nb_apply_results');
 END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named '`PROBABILITY`', which is the name generated by `APPLY`.

```
DECLARE
  v_accuracy    NUMBER;
     BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
                accuracy                     => v_accuracy,
                apply_result_table_name      => 'nb_apply_results',
                target_table_name            => 'mining_data_test_v',
                case_id_column_name          => 'cust_id',
                target_column_name           => 'affinity_card',
                confusion_matrix_table_name  => 'nb_confusion_matrix',
                score_column_name            => 'PREDICTION',
                score_criterion_column_name  => 'PROBABILITY',
                cost_matrix_table_name       => 'nb_cost_matrix',
                apply_result_schema_name     => null,
                target_schema_name           => null,
                cost_matrix_schema_name      => null,
                score_criterion_type         => 'COST');
       DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
    END;
    /
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```
**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

Compute a Confusion Matrix Based on Embedded Costs

You can use the `ADD_COST_MATRIX` procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN    DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample',
'nb_cost_matrix');END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
         SELECT cust_id,
              PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
              PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
          FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
   v_accuracy         NUMBER;
   BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
            accuracy                     => v_accuracy,
            apply_result_table_name      => 'nb_apply_results',
            target_table_name            => 'mining_data_test_v',
            case_id_column_name          => 'cust_id',
            target_column_name           => 'affinity_card',
            confusion_matrix_table_name  => 'nb_confusion_matrix',
            score_column_name            => 'PREDICTION',
            score_criterion_column_name  => 'COST',
            cost_matrix_table_name       => null,
            apply_result_schema_name     => null,
            target_schema_name           => null,
            cost_matrix_schema_name      => null,
            score_criterion_type         => 'COST');
   END;
   /
```

The results are:

```
**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

42.1.8.6 COMPUTE_CONFUSION_MATRIX_PART Procedure

The `COMPUTE_CONFUSION_MATRIX_PART` procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

`COMPUTE_CONFUSION_MATRIX_PART` provides support to computation of evaluation metrics per-partition for partitioned models. For non-partitioned models, refer to COMPUTE_CONFUSION_MATRIX Procedure.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

`COMPUTE_CONFUSION_MATRIX_PART` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about confusion matrixes and other test metrics for classification
>
> "COMPUTE_LIFT_PART Procedure"
>
> "COMPUTE_ROC_PART Procedure"

Syntax

```
DBMS_DATA_MINING.compute_confusion_matrix_part(
      accuracy                    OUT DM_NESTED_NUMERICALS,
      apply_result_table_name     IN  VARCHAR2,
      target_table_name           IN  VARCHAR2,
      case_id_column_name         IN  VARCHAR2,
      target_column_name          IN  VARCHAR2,
      confusion_matrix_table_name IN  VARCHAR2,
      score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
      cost_matrix_table_name      IN  VARCHAR2 DEFAULT NULL,
      apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
      target_schema_name          IN  VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name     IN  VARCHAR2 DEFAULT NULL,
      score_criterion_type        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-57 COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

Parameter	Description
`accuracy`	Output parameter containing the overall percentage accuracy of the predictions The output argument is changed from `NUMBER` to `DM_NESTED_NUMERICALS`
`apply_result_table_name`	Table containing the predictions
`target_table_name`	Table containing the known target values from the test data
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`confusion_matrix_table_name`	Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.

Table 42-57 (Cont.) COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

Parameter	Description
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`score_partition_column_name`	(Optional) Parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `COSTS`, the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table. If null, then the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, then the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided. If null, then the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `PROBABILITY`. To use costs as the scoring criterion, specify `COST`. If `score_criterion_type` is set to `COST` but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_CONFUSION_MATRIX_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to `COMPUTE_CONFUSION_MATRIX_PART`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_CONFUSION_MATRIX_PART` are in a table or view specified in `apply_result_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name           VARCHAR2,
            score_column_name             VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 42-55.

Table 42-58   Columns in a Cost Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`cost`	`BINARY_DOUBLE`

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for valid target data types
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The confusion matrix created by `COMPUTE_CONFUSION_MATRIX_PART` has the columns described in Table 42-56.

Table 42-59   Columns in a Confusion Matrix Part

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.

Table 42-59 (Cont.) Columns in a Confusion Matrix Part

Column Name	Data Type
value	BINARY_DOUBLE

See Also:

Oracle Machine Learning for SQL Concepts for more information about confusion matrixes

Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`.

Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
       SELECT cust_id,
              PREDICTION(nb_sh_clas_sample USING *) prediction,
              PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
       FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```
DECLARE
   v_accuracy    NUMBER;
      BEGIN
        DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
                   accuracy                     => v_accuracy,
                   apply_result_table_name      => 'nb_apply_results',
                   target_table_name            => 'mining_data_test_v',
                   case_id_column_name          => 'cust_id',
                   target_column_name           => 'affinity_card',
                   confusion_matrix_table_name  => 'nb_confusion_matrix',
                   score_column_name            => 'PREDICTION',
                   score_criterion_column_name  => 'PROBABILITY'
                   score_partition_column_name  => 'PARTITION_NAME'
                   cost_matrix_table_name       =>  null,
                   apply_result_schema_name     =>  null,
                   target_schema_name           =>  null,
                   cost_matrix_schema_name      =>  null,
                   score_criterion_type         => 'PROBABILITY');
        DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
      END;
      /
```

The confusion matrix and model accuracy are shown as follows.

```
 **** MODEL ACCURACY ****: .7847

SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         60
```

```
                   0                       0        891
                   1                       1        286
                   0                       1        263
```

Compute a Confusion Matrix Based on a Cost Matrix Table

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```
 SELECT * from NB_COST_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1        .75
                  1                      0        .25
                  1                      1          0
```

This statement shows how to generate the predictions using `APPLY`.

```
BEGIN
    DBMS_DATA_MINING.APPLY(
          model_name          => 'nb_sh_clas_sample',
          data_table_name     => 'mining_data_test_v',
          case_id_column_name => 'cust_id',
          result_table_name   => 'nb_apply_results');
 END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named '`PROBABILITY`', which is the name generated by `APPLY`.

```
DECLARE
  v_accuracy    NUMBER;
     BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
                accuracy                     => v_accuracy,
                apply_result_table_name      => 'nb_apply_results',
                target_table_name            => 'mining_data_test_v',
                case_id_column_name          => 'cust_id',
                target_column_name           => 'affinity_card',
                confusion_matrix_table_name  => 'nb_confusion_matrix',
                score_column_name            => 'PREDICTION',
                score_criterion_column_name  => 'PROBABILITY',
                score_partition_column_name  => 'PARTITION_NAME'
                cost_matrix_table_name       => 'nb_cost_matrix',
                apply_result_schema_name     => null,
                target_schema_name           => null,
                cost_matrix_schema_name      => null,
                score_criterion_type         => 'COST');
       DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
    END;
    /
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```
**** MODEL ACCURACY ****: .798

 SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

Compute a Confusion Matrix Based on Embedded Costs

You can use the ADD_COST_MATRIX procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN
DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample', 'nb_cost_matrix');
END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
         SELECT cust_id,
              PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
              PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
          FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
   v_accuracy         NUMBER;
   BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
            accuracy                     => v_accuracy,
            apply_result_table_name      => 'nb_apply_results',
            target_table_name            => 'mining_data_test_v',
            case_id_column_name          => 'cust_id',
            target_column_name           => 'affinity_card',
            confusion_matrix_table_name  => 'nb_confusion_matrix',
            score_column_name            => 'PREDICTION',
            score_criterion_column_name  => 'COST',
            score_partition_column_name  => 'PARTITION_NAME'
            cost_matrix_table_name       => null,
            apply_result_schema_name     => null,
            target_schema_name           => null,
            cost_matrix_schema_name      => null,
            score_criterion_type         => 'COST');
   END;
   /
```

The results are:

```
**** MODEL ACCURACY ****: .798

 SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
```

```
                    1                       1        255
                    0                       1        212
```

42.1.8.7 COMPUTE_LIFT Procedure

This procedure computes lift and stores the results in a table in the user's schema.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. `COMPUTE_LIFT` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about lift and test metrics for classification
>
> "COMPUTE_CONFUSION_MATRIX Procedure"
>
> "COMPUTE_ROC Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT (
      apply_result_table_name      IN VARCHAR2,
      target_table_name            IN VARCHAR2,
      case_id_column_name          IN VARCHAR2,
      target_column_name           IN VARCHAR2,
      lift_table_name              IN VARCHAR2,
```

```
      positive_target_value        IN VARCHAR2,
      score_column_name            IN VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN VARCHAR2 DEFAULT 'PROBABILITY',
      num_quantiles                IN NUMBER DEFAULT 10,
      cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN VARCHAR2 DEFAULT NULL,
      target_schema_name           IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN VARCHAR2 DEFAULT NULL
      score_criterion_type         IN VARCHAR2 DEFAULT 'PROBABILITY');
```

Parameters

Table 42-60 COMPUTE_LIFT Procedure Parameters

Parameter	Description
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`lift_table_name`	Table containing the lift statistics. The table will be created by the procedure in the user's schema. The columns in the lift table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate lift. If the target column is a `NUMBER`, you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is '`PREDICTION`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is '`PROBABILITY`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`num_quantiles`	Number of quantiles to be used in calculating lift. The default is 10.

Table 42-60 (Cont.) COMPUTE_LIFT Procedure Parameters

Parameter	Description
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to '`COST`', the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table. If null, the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is '`PROBABILITY`'. To use costs as the scoring criterion, specify '`COST`'. If `score_criterion_type` is set to '`COST`' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_LIFT` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the lift.
- Instead of passing a cost matrix to `COMPUTE_LIFT`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_LIFT` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name             VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 42-61.

Table 42-61 Columns in a Cost Matrix

Column Name	Data Type
actual_target_value	Type of the target column in the build data
predicted_target_value	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
cost	NUMBER

See Also:

Oracle Machine Learning for SQL Concepts for more information about cost matrixes

- The table created by `COMPUTE_LIFT` has the columns described in Table 42-62

Table 42-62 Columns in a Lift Table

Column Name	Data Type
quantile_number	NUMBER
probability_threshold	NUMBER
gain_cumulative	NUMBER
quantile_total_count	NUMBER
quantile_target_count	NUMBER
percent_records_cumulative	NUMBER
lift_cumulative	NUMBER
target_density_cumulative	NUMBER
targets_cumulative	NUMBER
non_targets_cumulative	NUMBER
lift_quantile	NUMBER
target_density	NUMBER

See Also:

Oracle Machine Learning for SQL Concepts for details about the information in the lift table

- When a cost matrix is passed to `COMPUTE_LIFT`, the cost threshold is returned in the `probability_threshold` column of the lift table.

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The example illustrates lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute lift as follows.

```
BEGIN
       DBMS_DATA_MINING.COMPUTE_LIFT (
            apply_result_table_name           => 'nb_apply_results',
            target_table_name                 => 'mining_data_test_v',
            case_id_column_name               => 'cust_id',
            target_column_name                => 'affinity_card',
            lift_table_name                       => 'nb_lift',
            positive_target_value             =>  to_char(1),
            score_column_name                 => 'PREDICTION',
            score_criterion_column_name    => 'PROBABILITY',
            num_quantiles                         =>  10,
            cost_matrix_table_name            =>  null,
            apply_result_schema_name          =>  null,
            target_schema_name                =>  null,
            cost_matrix_schema_name           =>  null,
            score_criterion_type              =>  'PROBABILITY');
      END;
      /
```

This query displays some of the statistics from the resulting lift table.

```
SQL>SELECT quantile_number, probability_threshold, gain_cumulative,
           quantile_total_count
           FROM nb_lift;

QUANTILE_NUMBER PROBABILITY_THRESHOLD GAIN_CUMULATIVE QUANTILE_TOTAL_COUNT
--------------- --------------------- --------------- --------------------
              1            .989335775       .15034965                   55
              2            .980534911       .26048951                   55
              3            .968506098      .374125874                   55
              4            .958975196      .493006993                   55
              5            .946705997      .587412587                   55
              6            .927454174       .66958042                   55
              7            .904403627      .748251748                   55
              8            .836482525      .839160839                   55
             10            .500184953               1                   54
```

42.1.8.8 COMPUTE_LIFT_PART Procedure

The `COMPUTE_LIFT_PART` procedure computes lift and stores the results in a table in the user's schema. This procedure provides support to the computation of evaluation metrics per-partition for partitioned models.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. `COMPUTE_LIFT_PART` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT_PART` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT_PART` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

See Also:

Oracle Machine Learning for SQL Concepts for more details about Lift and test metrics for classification

"COMPUTE_LIFT Procedure"

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_CONFUSION_MATRIX_PART Procedure"

"COMPUTE_ROC Procedure"

"COMPUTE_ROC_PART Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT_PART (
      apply_result_table_name     IN VARCHAR2,
      target_table_name           IN VARCHAR2,
      case_id_column_name         IN VARCHAR2,
      target_column_name          IN VARCHAR2,
      lift_table_name             IN VARCHAR2,
      positive_target_value       IN VARCHAR2,
      score_column_name           IN VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name IN VARCHAR2 DEFAULT 'PARTITION_NAME',
      num_quantiles               IN NUMBER   DEFAULT 10,
      cost_matrix_table_name      IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name    IN VARCHAR2 DEFAULT NULL,
      target_schema_name          IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name     IN VARCHAR2 DEFAULT NULL,
      score_criterion_type        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-63 COMPUTE_LIFT_PART Procedure Parameters

Parameter	Description
`apply_result_table_name`	Table containing the predictions
`target_table_name`	Table containing the known target values from the test data
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`lift_table_name`	Table containing the Lift statistics. The table will be created by the procedure in the user's schema. The columns in the Lift table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate Lift. If the target column is a `NUMBER`, then you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`score_partition_column_name`	Optional parameter indicating the column containing the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`num_quantiles`	Number of quantiles to be used in calculating Lift. The default is 10.
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `COST`, then the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.

Table 42-63 (Cont.) COMPUTE_LIFT_PART Procedure Parameters

Parameter	Description
`apply_result_schema_name`	Schema of the apply results table If null, then the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets If null, then the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided If null, then the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `PROBABILITY`. To use costs as the scoring criterion, specify `COST`. If `score_criterion_type` is set to `COST` but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_LIFT_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the Lift.
- Instead of passing a cost matrix to `COMPUTE_LIFT_PART`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_LIFT_PART` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 42-61.

Table 42-64 Columns in a Cost Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.

Table 42-64 (Cont.) Columns in a Cost Matrix

Column Name	Data Type
cost	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The table created by `COMPUTE_LIFT_PART` has the columns described in Table 42-62

Table 42-65 Columns in a COMPUTE_LIFT_PART Table

Column Name	Data Type
quantile_number	NUMBER
probability_threshold	NUMBER
gain_cumulative	NUMBER
quantile_total_count	NUMBER
quantile_target_count	NUMBER
percent_records_cumulative	NUMBER
lift_cumulative	NUMBER
target_density_cumulative	NUMBER
targets_cumulative	NUMBER
non_targets_cumulative	NUMBER
lift_quantile	NUMBER
target_density	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for details about the information in the Lift table

- When a cost matrix is passed to `COMPUTE_LIFT_PART`, the cost threshold is returned in the `probability_threshold` column of the Lift table.

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The example illustrates Lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".

For a partitioned model example, see "COMPUTE_CONFUSION_MATRIX_PART Procedure".

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute Lift as follows.

```
BEGIN
      DBMS_DATA_MINING.COMPUTE_LIFT_PART (
             apply_result_table_name     => 'nb_apply_results',
             target_table_name           => 'mining_data_test_v',
             case_id_column_name         => 'cust_id',
             target_column_name          => 'affinity_card',
             lift_table_name             => 'nb_lift',
             positive_target_value       =>  to_char(1),
             score_column_name           => 'PREDICTION',
             score_criterion_column_name => 'PROBABILITY',
             score_partition_column_name => 'PARTITITON_NAME',
             num_quantiles               =>  10,
             cost_matrix_table_name      =>  null,
             apply_result_schema_name    =>  null,
             target_schema_name          =>  null,
             cost_matrix_schema_name     =>  null,
             score_criterion_type        =>  'PROBABILITY');
END;
/
```

This query displays some of the statistics from the resulting Lift table.

```
SELECT quantile_number, probability_threshold, gain_cumulative,
           quantile_total_count
           FROM nb_lift;

QUANTILE_NUMBER PROBABILITY_THRESHOLD GAIN_CUMULATIVE QUANTILE_TOTAL_COUNT
--------------- --------------------- --------------- --------------------
              1            .989335775       .15034965                   55
              2            .980534911       .26048951                   55
              3            .968506098      .374125874                   55
              4            .958975196      .493006993                   55
              5            .946705997      .587412587                   55
              6            .927454174       .66958042                   55
              7            .904403627      .748251748                   55
              8            .836482525      .839160839                   55
             10            .500184953               1                   54
```

42.1.8.9 COMPUTE_ROC Procedure

This procedure computes the receiver operating characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. `COMPUTE_ROC` compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is 0.5. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an X-Y axis. The false positive rate is placed on the X axis. The true positive rate is placed on the Y axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

`COMPUTE_ROC` accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about ROC and test metrics for classification
>
> "COMPUTE_CONFUSION_MATRIX Procedure"
>
> "COMPUTE_LIFT Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_ROC (
      roc_area_under_curve         OUT NUMBER,
      apply_result_table_name      IN  VARCHAR2,
      target_table_name            IN  VARCHAR2,
      case_id_column_name          IN  VARCHAR2,
      target_column_name           IN  VARCHAR2,
      roc_table_name               IN  VARCHAR2,
      positive_target_value        IN  VARCHAR2,
      score_column_name            IN  VARCHAR2 DEFAULT 'PREDICTION',
```

```
score_criterion_column_name  IN  VARCHAR2 DEFAULT 'PROBABILITY',
apply_result_schema_name     IN  VARCHAR2 DEFAULT NULL,
target_schema_name           IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-66 COMPUTE_ROC Procedure Parameters

Parameter	Description
`roc_area_under_the_curve`	Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another.
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`roc_table_name`	Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a `NUMBER`, you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is '`PREDICTION`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is '`PROBABILITY`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`apply_result_schema_name`	Schema of the apply results table. If null, the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, the user's schema is assumed.

Usage Notes

- The predictive information you pass to `COMPUTE_ROC` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.

- The predictions that you pass to `COMPUTE_ROC` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- The table created by `COMPUTE_ROC` has the columns shown in Table 42-67.

Table 42-67 COMPUTE_ROC Output

Column	Datatype
`probability`	`BINARY_DOUBLE`
`true_positives`	`NUMBER`
`false_negatives`	`NUMBER`
`false_positives`	`NUMBER`
`true_negatives`	`NUMBER`
`true_positive_fraction`	`NUMBER`
`false_positive_fraction`	`NUMBER`

See Also:

Oracle Machine Learning for SQL Concepts for details about the output of `COMPUTE_ROC`

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

  Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```
SELECT case_id_column_name
       FROM apply_result_table_name
       WHERE probability > probability_threshold
       ORDER BY probability DESC;
```

- There are two approaches to identifying the most desirable probability threshold. Which approach you use depends on whether or not you know the relative cost of positive versus negative class prediction errors.

  If the costs are known, you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query like this.

```
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false_positives cost
    FROM ROC_table
  GROUP BY probability_threshold),
    minCost AS (
      SELECT min(cost) minCost
        FROM cost)
      SELECT max(probability_threshold)probability_threshold
        FROM cost, minCost
    WHERE cost = minCost;
```

If relative costs are not well known, you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```
SELECT * FROM ROC_table
         ORDER BY probability_threshold;
```

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```
DECLARE
     v_area_under_curve NUMBER;
BEGIN
     DBMS_DATA_MINING.COMPUTE_ROC (
         roc_area_under_curve        => v_area_under_curve,
         apply_result_table_name     => 'nb_apply_results',
         target_table_name           => 'mining_data_test_v',
         case_id_column_name         => 'cust_id',
         target_column_name          => 'mining_data_test_v',
         roc_table_name              => 'nb_roc',
         positive_target_value       => '1',
         score_column_name           => 'PREDICTION',
         score_criterion_column_name => 'PROBABILITY');
     DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
     ROUND(v_area_under_curve,4));
END;
/
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212

 SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION
           FROM NB_ROC;
```

```
PROBABILITY  TRUE_POSITIVE_FRACTION  FALSE_POSITIVE_FRACTION
-----------  ----------------------  -----------------------
     .00000                       1                        1
     .50018               .826589595               .227902946
     .53851               .823699422               .221837088
     .54991               .820809249               .217504333
     .55628               .815028902               .215771231
     .55628               .817919075               .215771231
     .57563               .800578035               .214904679
     .57563               .812138728               .214904679
      .                    .                        .
      .                    .                        .
      .                    .                        .
```

42.1.8.10 COMPUTE_ROC_PART Procedure

The `COMPUTE_ROC_PART` procedure computes Receiver Operating Characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy. This procedure provides support to computation of evaluation metrics per-partition for partitioned models.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. `COMPUTE_ROC_PART` compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is `0.5`. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an x-y axis. The false positive rate is placed on the x-axis. The true positive rate is placed on the y-axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

`COMPUTE_ROC_PART` accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

See Also:

Oracle Machine Learning for SQL Concepts for more details about ROC and test metrics for Classification

"COMPUTE_ROC Procedure"

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_LIFT_PART Procedure"

"COMPUTE_LIFT Procedure"

Syntax

```
DBMS_DATA_MINING.compute_roc_part(
      roc_area_under_curve        OUT DM_NESTED_NUMERICALS,
      apply_result_table_name     IN  VARCHAR2,
      target_table_name           IN  VARCHAR2,
      case_id_column_name         IN  VARCHAR2,
      target_column_name          IN  VARCHAR2,
      roc_table_name              IN  VARCHAR2,
      positive_target_value       IN  VARCHAR2,
      score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
      apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
      target_schema_name          IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-68 COMPUTE_ROC_PART Procedure Parameters

Parameter	Description
`roc_area_under_the_curve`	Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another. The output argument is changed from `NUMBER` to `DM_NESTED_NUMERICALS`.
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.

Table 42-68 (Cont.) COMPUTE_ROC_PART Procedure Parameters

Parameter	Description
`roc_table_name`	Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a `NUMBER`, then you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_partition_column_name`	Optional parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`apply_result_schema_name`	Schema of the apply results table. If null, then the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, then the user's schema is assumed.

Usage Notes

- The predictive information you pass to `COMPUTE_ROC_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.
- The predictions that you pass to `COMPUTE_ROC_PART` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- The `COMPUTE_ROC_PART` table has the following columns:

Table 42-69 COMPUTE_ROC_PART Output

Column	Data Type
`probability`	`BINARY_DOUBLE`

Table 42-69 (Cont.) COMPUTE_ROC_PART Output

Column	Data Type
true_positives	NUMBER
false_negatives	NUMBER
false_positives	NUMBER
true_negatives	NUMBER
true_positive_fraction	NUMBER
false_positive_fraction	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for details about the output of `COMPUTE_ROC_PART`

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

  Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```
SELECT case_id_column_name
       FROM apply_result_table_name
       WHERE probability > probability_threshold
       ORDER BY probability DESC;
```

- There are two approaches to identify the most desirable probability threshold. The approach you use depends on whether you know the relative cost of positive versus negative class prediction errors.

  If the costs are known, then you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query as follows:

```
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false_positives cost
    FROM ROC_table
  GROUP BY probability_threshold),
    minCost AS (
      SELECT min(cost) minCost
        FROM cost)
      SELECT max(probability_threshold)probability_threshold
        FROM cost, minCost
    WHERE cost = minCost;
```

  If relative costs are not well known, then you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```
SELECT * FROM ROC_table
        ORDER BY probability_threshold;
```

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```
DECLARE
     v_area_under_curve NUMBER;
BEGIN
     DBMS_DATA_MINING.COMPUTE_ROC_PART (
         roc_area_under_curve        => v_area_under_curve,
         apply_result_table_name     => 'nb_apply_results',
         target_table_name           => 'mining_data_test_v',
         case_id_column_name         => 'cust_id',
         target_column_name          => 'affinity_card',
         roc_table_name              => 'nb_roc',
         positive_target_value       => '1',
         score_column_name           => 'PREDICTION',
         score_criterion_column_name => 'PROBABILITY');
         score_partition_column_name => 'PARTITION_NAME'
     DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
     ROUND(v_area_under_curve,4));
END;
/
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212

 SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION
           FROM NB_ROC;

PROBABILITY  TRUE_POSITIVE_FRACTION  FALSE_POSITIVE_FRACTION
-----------  ----------------------  -----------------------
     .00000                       1                        1
     .50018              .826589595               .227902946
     .53851              .823699422               .221837088
     .54991              .820809249               .217504333
     .55628              .815028902               .215771231
     .55628              .817919075               .215771231
     .57563              .800578035               .214904679
     .57563              .812138728               .214904679
      .                   .                        .
      .                   .                        .
      .                   .                        .
```

42.1.8.11 CREATE_MODEL Procedure

This procedure creates an Oracle Machine Learning for SQL model with a given machine learning function.

Syntax

```
DBMS_DATA_MINING.CREATE_MODEL (
      model_name            IN VARCHAR2,
      mining_function       IN VARCHAR2,
      data_table_name       IN VARCHAR2,
      case_id_column_name   IN VARCHAR2,
      target_column_name    IN VARCHAR2 DEFAULT NULL,
      settings_table_name   IN VARCHAR2 DEFAULT NULL,
      data_schema_name      IN VARCHAR2 DEFAULT NULL,
      settings_schema_name  IN VARCHAR2 DEFAULT NULL,
      xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

Parameters

Table 42-70 CREATE_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used. See the Usage Notes for model naming restrictions.
`mining_function`	The machine learning function. Values are listed in Table 42-3.
`data_table_name`	Table or view containing the build data
`case_id_column_name`	Case identifier column in the build data.
`target_column_name`	For supervised models, the target column in the build data. `NULL` for unsupervised models.
`settings_table_name`	Table containing build settings for the model. `NULL` if there is no settings table (only default settings are used).
`data_schema_name`	Schema hosting the build data. If `NULL`, then the user's schema is assumed.
`settings_schema_name`	Schema hosting the settings table. If `NULL`then the user's schema is assumed.

Table 42-70 (Cont.) CREATE_MODEL Procedure Parameters

Parameter	Description

`xform_list`	A list of transformations to be used in addition to or instead of automatic transformations, depending on the value of the `PREP_AUTO` setting. (See "Automatic Data Preparation".)
	The datatype of `xform_list` is `TRANSFORM_LIST`, which consists of records of type `TRANSFORM_REC`. Each `TRANSFORM_REC` specifies the transformation information for a single attribute.
	`TYPE`
	`  TRANFORM_REC      IS RECORD (`
	`     attribute_name        VARCHAR2(4000),`
	`     attribute_subname     VARCHAR2(4000),`
	`     expression            EXPRESSION_REC,`
	`     reverse_expression    EXPRESSION_REC,`
	`     attribute_spec        VARCHAR2(4000));`
	The `expression` field stores a SQL expression for transforming the attribute. The `reverse_expression` field stores a SQL expression for reversing the transformation in model details and, if the attribute is a target, in the results of scoring. The SQL expressions are manipulated by routines in the `DBMS_DATA_MINING_TRANSFORM` package:
	• SET_EXPRESSION Procedure
	• GET_EXPRESSION Function
	• SET_TRANSFORM Procedure
	The `attribute_spec` field identifies individualized treatment for the attribute. See the Usage Notes for details.
	See Table 42-123for details about the `TRANSFORM_REC` type.

Usage Notes

1. You can use the `attribute_spec` field of the `xform_list` argument to identify an attribute as unstructured text or to disable Automatic Data Preparation for the attribute. The `attribute_spec` can have the following values:

   - `TEXT`: Indicates that the attribute contains unstructured text. The `TEXT` value may optionally be followed by `POLICY_NAME`, `TOKEN_TYPE`, `MAX_FEATURES`, and `MIN_DOCUMENTS` parameters.

     `TOKEN_TYPE` has the following possible values: `NORMAL`, `STEM`, `THEME`, `SYNONYM`, `BIGRAM`, `STEM_BIGRAM`. `SYNONYM` may be optionally followed by a thesaurus name in square brackets.

     `MAX_FEATURES` specifies the maximum number of tokens extracted from the text.

     `MIN_DOCUMENTS` specifies the minimal number of documents in which every selected token shall occur. (For information about creating a text policy, see `CTX_DDL.CREATE_POLICY` in *Oracle Text Reference*).

     Oracle Machine Learning for SQL can process columns of `VARCHAR2`/`CHAR`, `CLOB`, `BLOB`, and `BFILE` as text. If the column is `VARCHAR2` or `CHAR` and you do not specify `TEXT`, then OML4SQL processes the column as categorical data. If the column is `CLOB`, then OML4SQL processes it as text by default (You do not need to specify it as `TEXT`. However, you do need to provide an Oracle Text

Policy in the settings). If the column is `BLOB` or `BFILE`, then you must specify it as `TEXT`, otherwise `CREATE_MODEL` returns an error.

If you specify `TEXT` for a nested column or for an attribute in a nested column, then `CREATE_MODEL` returns an error.

- `NOPREP`: Disables ADP for the attribute. When ADP is `OFF`, the `NOPREP` value is ignored.

  You can specify `NOPREP` for a nested column, but not for an attribute in a nested column. If you specify `NOPREP` for an attribute in a nested column when ADP is on, then `CREATE_MODEL` will return an error.

2. You can obtain information about a model by querying the Data Dictionary views.

```
ALL/USER/DBA_MINING_MODELS
ALL/USER/DBA_MINING_MODEL_ATTRIBUTES
ALL/USER/DBA_MINING_MODEL_SETTINGS
ALL/USER/DBA_MINING_MODEL_VIEWS
ALL/USER/DBA_MINING_MODEL_PARTITIONS
ALL/USER/DBA_MINING_MODEL_XFORMS
```

   You can obtain information about model attributes by querying the model details through model views. Refer to *Oracle Machine Learning for SQL User's Guide*.

3. The naming rules for models are more restrictive than the naming rules for most database schema objects. A model name must satisfy the following additional requirements:

   - It must be 123 or fewer characters long.
   - It must be a nonquoted identifier. Oracle requires that nonquoted identifiers contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#); the initial character must be alphabetic. Oracle strongly discourages the use of the dollar sign and pound sign in nonquoted literals.

   Naming requirements for schema objects are fully documented in *Oracle Database SQL Language Reference*.

4. To build a partitioned model, you must provide additional settings.

   The setting for partitioning columns are as follows:

```
INSERT INTO settings_table VALUES ('ODMS_PARTITION_COLUMNS', 'GENDER,
AGE');
```

   To set user-defined partition number for a model, the setting is as follows:

```
INSERT INTO settings_table VALUES ('ODMS_MAX_PARTITIONS', '10');
```

   The default value for maximum number of partitions is `1000`.

5. By passing an `xform_list` to `CREATE_MODEL`, you can specify a list of transformations to be performed on the input data. If the `PREP_AUTO` setting is `ON`, the transformations are used in addition to the automatic transformations. If the `PREP_AUTO` setting is `OFF`, the specified transformations are the only ones implemented by the model. In both cases, transformation definitions are embedded in the model and run automatically whenever the model is applied. See "Automatic Data Preparation". Other transforms that can be

specified with `xform_list` include `FORCE_IN`. Refer to *Oracle Machine Learning for SQL User's Guide*.

Examples

The first example builds a classification model using the Support Vector Machine algorithm.

```
-- Create the settings table
CREATE TABLE svm_model_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(30));

-- Populate the settings table
-- Specify SVM. By default, Naive Bayes is used for classification.
-- Specify ADP. By default, ADP is not used.
BEGIN
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
  COMMIT;
END;
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svm_model_settings');
END;
/
```

You can display the model settings with the following query:

```
SELECT * FROM user_mining_model_settings
       WHERE model_name IN 'SVM_MODEL';

MODEL_NAME     SETTING_NAME           SETTING_VALUE                 SETTING
-------------  ---------------------- ----------------------------- -------
SVM_MODEL      ALGO_NAME              ALGO_SUPPORT_VECTOR_MACHINES  INPUT

SVM_MODEL      SVMS_STD_DEV           3.004524                      DEFAULT
SVM_MODEL      PREP_AUTO              ON                            INPUT
SVM_MODEL      SVMS_COMPLEXITY_FACTOR 1.887389                      DEFAULT
SVM_MODEL      SVMS_KERNEL_FUNCTION   SVMS_LINEAR                      DEFAULT
SVM_MODEL      SVMS_CONV_TOLERANCE    .001                          DEFAULT
```

The following is an example of querying a model view instead of the older `GEL_MODEL_DETAILS_SVM` routine.

```
SELECT target_value, attribute_name, attribute_value, coefficient
  FROM DM$VLSVM_MODEL;
```

The second example creates an anomaly detection model. Anomaly detection uses SVM classification without a target. This example uses the same settings table created for the SVM classification model in the first example.

```
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'anomaly_detect_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => null,
    settings_table_name => 'svm_model_settings');
END;
/
```

This query shows that the models created in these examples are the only ones in your schema.

```
SELECT model_name, mining_function, algorithm FROM user_mining_models;

MODEL_NAME                MINING_FUNCTION      ALGORITHM
---------------------- -------------------- ------------------------------
SVM_MODEL                 CLASSIFICATION       SUPPORT_VECTOR_MACHINES
ANOMALY_DETECT_MODEL      CLASSIFICATION       SUPPORT_VECTOR_MACHINES
```

This query shows that only the SVM classification model has a target.

```
SELECT model_name, attribute_name, attribute_type, target
       FROM user_mining_model_attributes
       WHERE target = 'YES';

MODEL_NAME          ATTRIBUTE_NAME   ATTRIBUTE_TYPE     TARGET
------------------ --------------- ----------------- ------
SVM_MODEL           AFFINITY_CARD    CATEGORICAL         YES
```

42.1.8.12 CREATE_MODEL2 Procedure

The `CREATE_MODEL2` procedure is an alternate procedure to the `CREATE_MODEL` procedure, which enables creating a model without extra persistence stages. In the `CREATE_MODEL` procedure, the input is a table or a view and if such an object is not already present, the user must create it. By using the `CREATE_MODEL2` procedure, the user does not need to create such transient database objects.

Syntax

```
DBMS_DATA_MINING.CREATE_MODEL2 (
     model_name             IN VARCHAR2,
     mining_function        IN VARCHAR2,
     data_query             IN CLOB,
     set_list               IN SETTING_LIST,
     case_id_column_name    IN VARCHAR2 DEFAULT NULL,
     target_column_name     IN VARCHAR2 DEFAULT NULL,
     xform_list             IN TRANSFORM_LIST DEFAULT NULL);
```

Parameters

Table 42-71 CREATE_MODEL2 Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [`schema_name`.]`model_name`. If you do not specify a schema, then the current schema is used. See the Usage Notes, CREATE_MODEL Procedure for model naming restrictions.
`mining_function`	The machine learning function. Values are listed in DBMS_DATA_MINING — Machine Learning Function Settings.
`data_query`	A query which provides training data for building the model.
`set_list`	Specifies the `SETTING_LIST` `SETTING_LIST` is a table of CLOB index by `VARCHAR2(30);` Where the index is the setting name and the CLOB is the setting value for that name.
`case_id_column_name`	Case identifier column in the build data.
`target_column_name`	For supervised models, the target column in the build data. `NULL` for unsupervised models.
`xform_list`	Refer to CREATE_MODEL Procedure.

Usage Notes

Refer to CREATE_MODEL Procedure for Usage Notes.

Examples

The following example uses the Support Vector Machine algorithm.

```
declare
 v_setlst DBMS_DATA_MINING.SETTING_LIST;

BEGIN
  v_setlst(dbms_data_mining.algo_name) :=
dbms_data_mining.algo_support_vector_machines;
  v_setlst(dbms_data_mining.prep_auto) :=
dbms_data_mining.prep_auto_on;

DBMS_DATA_MINING.CREATE_MODEL2(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_query          => 'select * from mining_data_build_v',
    data_table_name     => 'mining_data_build_v',
    case_id_column_name=> 'cust_id',
    target_column_name => 'affinity_card',
    set_list            => v_setlst,
    case_id_column_name=> 'cust_id',
    target_column_name => 'affinity_card');
END;
/
```

42.1.8.13 Create Model Using Registration Information

Create model function fetches the setting information from JSON object.

Usage Notes

If an algorithm is registered, user can create model using the registered algorithm name. Since all R scripts and default setting values are already registered, providing the value through the setting table is not necessary. This makes the use of this algorithm easier.

Examples

The first example builds a Classification model using the GLM algorithm.

```
CREATE TABLE GLM_RDEMO_SETTINGS_CL (

   setting_name  VARCHAR2(30),
   setting_value VARCHAR2(4000));
   BEGIN
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         ('ALGO_EXTENSIBLE_LANG', 'R');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         (dbms_data_mining.ralg_registration_algo_name, 't1');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
        (dbms_data_mining.odms_formula,
        'AGE + EDUCATION + HOUSEHOLD_SIZE + OCCUPATION');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         ('RALG_PARAMETER_FAMILY',   'binomial(logit)' );
   END;
   /
     BEGIN
          DBMS_DATA_MINING.CREATE_MODEL(
          model_name                  =>    'GLM_RDEMO_CLASSIFICATION',
          mining_function             =>     dbms_data_mining.classification,
          data_table_name             =>    'mining_data_build_v',
          case_id_column_name         =>    'CUST_ID',
          target_column_name          =>    'AFFINITY_CARD',
          settings_table_name         =>    'GLM_RDEMO_SETTINGS_CL');
      END;
      /
```

42.1.8.14 DROP_ALGORITHM Procedure

This function is used to drop the registered algorithm information.

Syntax

```
DBMS_DATA_MINING.DROP_ALGORITHM (algorithm_name   IN  VARCHAR2(30),
                                 cascade          IN  BOOLEAN default FALSE)
```

Parameters

Table 42-72 DROP_ALGORITHM Procedure Parameters

Parameter	Description
`algorithm_name`	Name of the algorithm.
`cascade`	If the cascade option is `TRUE`, all the models with this algorithms are forced to drop. There after, the algorithm is dropped. The default value is `FALSE`.

Usage Note

- To drop a machine learning model, you must be the owner or you must have the `RQADMIN` privilege. See *Oracle Machine Learning for SQL User's Guide* for information about privileges for machine learning.
- Make sure a model is not built on the algorithm, then drop the algorithm from the system table.
- If you try to drop an algorithm with a model built on it, then an error is displayed.

42.1.8.15 DROP_PARTITION Procedure

Syntax

```
DBMS_DATA_MINING.DROP_PARTITION (
      model_name                 IN VARCHAR2,
      partition_name             IN VARCHAR2);
```

Parameters

Table 42-73 DROP_PARTITION Procedure Parameters

Parameters	Description
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Name of the partition that must be dropped.

42.1.8.16 DROP_MODEL Procedure

This procedure deletes the specified machine learning model.

Syntax

```
DBMS_DATA_MINING.DROP_MODEL (model_name IN VARCHAR2,
                             force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 42-74 DROP_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`force`	Forces the machine learning model to be dropped even if it is invalid. A machine learning model may be invalid if a serious system error interrupted the model build process.

Usage Note

To drop a machine learning model, you must be the owner or you must have the `DROP ANY MINING MODEL` privilege. See *Oracle Data Mining User's Guide* for information about privileges for Oracle Machine Learning for SQL.

Example

You can use the following command to delete a valid machine learning model named `nb_sh_clas_sample` that exists in your schema.

```
BEGIN
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'nb_sh_clas_sample');
END;
/
```

42.1.8.17 EXPORT_MODEL Procedure

This procedure exports the specified machine learning models to a dump file set.

To import the models from the dump file set, use the IMPORT_MODEL Procedure. `EXPORT_MODEL` and `IMPORT_MODEL` use Oracle Data Pump technology.

When Oracle Data Pump is used to export/import an entire schema or database, the machine learning models in the schema or database are included. However, `EXPORT_MODEL` and `IMPORT_MODEL` are the only utilities that support the export/import of individual models.

> **See Also:**
>
> *Oracle Database Utilities* for information about Oracle Data Pump
>
> *Oracle Machine Learning for SQL User's Guide* for more information about exporting and importing machine learning models

Syntax

```
DBMS_DATA_MINING.EXPORT_MODEL (
      filename          IN VARCHAR2,
      directory         IN VARCHAR2,
      model_filter      IN VARCHAR2 DEFAULT NULL,
      filesize          IN VARCHAR2 DEFAULT NULL,
      operation         IN VARCHAR2 DEFAULT NULL,
```

```
    remote_link        IN VARCHAR2 DEFAULT NULL,
    jobname            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-75 EXPORT_MODEL Procedure Parameters

Parameter	Description
`filename`	Name of the dump file set to which the models should be exported. The name must be unique within the schema. The dump file set can contain one or more files. The number of files in a dump file set is determined by the size of the models being exported (both metadata and data) and a specified or estimated maximum file size. You can specify the file size in the `filesize` parameter, or you can use the `operation` parameter to cause Oracle Data Pump to estimate the file size. If the size of the models to export is greater than the maximum file size, one or more additional files are created. When the export operation completes successfully, the name of the dump file set is automatically expanded to *`filename01.dmp`*, even if there is only one file in the dump set. If there are additional files, they are named sequentially as *`filename02.dmp`*, *`filename03.dmp`*, and so forth.
`directory`	Name of a pre-defined directory object that specifies where the dump file set should be created. The exporting user must have read/write privileges on the directory object and on the file system directory that it identifies. See *Oracle Database SQL Language Reference* for information about directory objects.
`model_filter`	Optional parameter that specifies which model or models to export. If you do not specify a value for `model_filter`, all models in the schema are exported. You can also specify `NULL` (the default) or `'ALL'` to export all models. You can export individual models by name and groups of models based on machine learning function or algorithm. For instance, you could export all regression models or all Naive Bayes models. Examples are provided in Table 42-76.
`filesize`	Optional parameter that specifies the maximum size of a file in the dump file set. The size may be specified in bytes, kilobytes (K), megabytes (M), or gigabytes (G). The default size is 50 MB. If the size of the models to export is larger than `filesize`, one or more additional files are created within the dump set. See the description of the `filename` parameter for more information.
`operation`	Optional parameter that specifies whether or not to estimate the size of the files in the dump set. By default the size is not estimated and the value of the `filesize` parameter determines the size of the files. You can specify either of the following values for `operation`: • `'EXPORT'` — Export all or the specified models. (Default) • `'ESTIMATE'` — Estimate the size of the exporting models.

Table 42-75 (Cont.) EXPORT_MODEL Procedure Parameters

Parameter	Description
`remote_link`	Optional parameter that specifies the name of a database link to a remote system. The default value is `NULL`. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for `remote_link`, you can export the models in the remote database. The `EXP_FULL_DATABASE` role is required for exporting the remote models. The `EXP_FULL_DATABASE` privilege, the `CREATE DATABASE LINK` privilege, and other privileges may also be required.
`jobname`	Optional parameter that specifies the name of the export job. By default, the name has the form *`username_exp_nnnn`*, where *`nnnn`* is a number. For example, a job name in the `SCOTT` schema might be `SCOTT_exp_134`. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the export job, named *`jobname.log`*, is created in the same directory as the dump file set.

Usage Notes

The `model_filter` parameter specifies which models to export. You can list the models by name, or you can specify all models that have the same machine learning function or algorithm. You can query the `USER_MINING_MODELS` view to list the models in your schema.

```
SQL> describe user_mining_models
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MODEL_NAME                                NOT NULL VARCHAR2(30)
 MINING_FUNCTION                                    VARCHAR2(30)
 ALGORITHM                                          VARCHAR2(30)
 CREATION_DATE                             NOT NULL DATE
 BUILD_DURATION                                     NUMBER
 MODEL_SIZE                                         NUMBER
 COMMENTS                                           VARCHAR2(4000)
```

Examples of model filters are provided in Table 42-76.

Table 42-76 Sample Values for the Model Filter Parameter

Sample Value	Meaning
`'mymodel'`	Export the model named `mymodel`
`'name= ''mymodel'''`	Export the model named `mymodel`
`'name IN (''mymodel2'',''mymodel3'')'`	Export the models named `mymodel2` and `mymodel3`
`'ALGORITHM_NAME = ''NAIVE_BAYES'''`	Export all Naive Bayes models. See Table 42-5 for a list of algorithm names.
`'FUNCTION_NAME =''CLASSIFICATION'''`	Export all classification models. See Table 42-3 for a list of machine learning functions.

Examples

1. The following statement exports all the models in the `oml_user3` schema to a dump file set called `models_out` in the directory `$ORACLE_HOME/rdbms/log`. This directory is mapped to a directory object called `DATA_PUMP_DIR`. The `oml_user3` user has read/write access to the directory and to the directory object.

   ```
   SQL>execute dbms_data_mining.export_model ('models_out', 'DATA_PUMP_DIR');
   ```

   You can exit SQL*Plus and list the resulting dump file and log file.

   ```
   SQL>EXIT
   >cd $ORACLE_HOME/rdbms/log
   >ls
   >oml_user3_exp_1027.log  models_out01.dmp
   ```

2. The following example uses the same directory object and is run by the same user. This example exports the models called `NMF_SH_SAMPLE` and `SVMR_SH_REGR_SAMPLE` to a different dump file set in the same directory.

   ```
   SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL ( 'models2_out', 'DATA_PUMP_DIR',
               'name in (''NMF_SH_SAMPLE'', ''SVMR_SH_REGR_SAMPLE'')');
   SQL>EXIT
   >cd $ORACLE_HOME/rdbms/log
   >ls
   >oml_user3_exp_1027.log  models_out01.dmp
    oml_user3_exp_924.log  models2_out01.dmp
   ```

3. The following examples show how to export models with specific algorithm and machine learning function names.

   ```
   SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('algo.dmp','DM_DUMP',
           'ALGORITHM_NAME IN (''O_CLUSTER'',''GENERALIZED_LINEAR_MODEL'',
           ''SUPPORT_VECTOR_MACHINES'',''NAIVE_BAYES'')');

   SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('func.dmp', 'DM_DUMP',
           'FUNCTION_NAME IN (CLASSIFICATION,CLUSTERING,FEATURE_EXTRACTION)');
   ```

42.1.8.18 EXPORT_SERMODEL Procedure

This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

When exporting a model in serialized format, the user must pass in an empty `BLOB` locator and specify the model name to be exported. If the model is partitioned, the user can optionally select an individual partition to export, otherwise all partitions are exported. The returned `BLOB` contains the content that can be deployed.

Syntax

```
DBMS_DATA_MINING.EXPORT_SERMODEL (
      model_data     IN OUT NOCOPY BLOB,
      model_name     IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-77 EXPORT_SERMODEL Procedure Parameters

Parameter	Description
`model_data`	Provides serialized model data.
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Name of the partition that must be exported.

Examples

The following statement exports all of the models in a serialized format.

```
DECLARE
 v_blob blob;
BEGIN
 dbms_lob.createtemporary(v_blob, FALSE);
 dbms_data_mining.export_sermodel(v_blob, 'MY_MODEL');
-- save v_blob somewhere (e.g., bfile, etc.)
 dbms_lob.freetemporary(v_blob);
END;
/
```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for more information about exporting and importing machine learning models

42.1.8.19 FETCH_JSON_SCHEMA Procedure

User can fetch and read JSON schema from the `ALL_MINING_ALGORITHMS` view. This function returns the pre-registered JSON schema for R extensible algorithms.

Syntax

```
DBMS_DATA_MINING.FETCH_JSON_SCHEMA RETURN CLOB;
```

Parameters

Table 42-78 FETCH_JSON_SCHEMA Procedure Parameters

Parameter	Description
`RETURN`	This function returns the pre-registered JSON schema for R extensibility. The default value is `CLOB`.

Usage Note

If a user wants to register a new algorithm using the algorithm registration function, they must fetch and follow the pre-registered JSON schema using this function, when they create the required JSON object metadata, and then pass it to the registration function.

42.1.8.20 GET_ASSOCIATION_RULES Function

The `GET_ASSOCIATION_RULES` function returns the rules produced by an association model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

You can specify filtering criteria to `GET_ASSOCIATION_RULES` to return a subset of the rules. Filtering criteria can improve the performance of the table function. If the number of rules is large, then the greatest performance improvement will result from specifying the `topn` parameter.

Syntax

```
DBMS_DATA_MINING.get_association_rules(
      model_name       IN VARCHAR2,
      topn             IN NUMBER DEFAULT NULL,
      rule_id          IN INTEGER DEFAULT NULL,
      min_confidence   IN NUMBER DEFAULT NULL,
      min_support      IN NUMBER DEFAULT NULL,
      max_rule_length  IN INTEGER DEFAULT NULL,
      min_rule_length  IN INTEGER DEFAULT NULL,
      sort_order       IN ORA_MINING_VARCHAR2_NT DEFAULT NULL,
      antecedent_items IN DM_ITEMS DEFAULT NULL,
      consequent_items IN DM_ITEMS DEFAULT NULL,
      min_lift         IN NUMBER DEFAULT NULL,
      partition_name   IN VARCHAR2 DEFAULT NULL)
  RETURN DM_Rules PIPELINED;
```

Parameters

Table 42-79 GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used. This is the only required parameter of `GET_ASSOCIATION_RULES`. All other parameters specify optional filters on the rules to return.
`topn`	Returns the *n* top rules ordered by confidence and then support, both descending. If you specify a sort order, then the top *n* rules are derived after the sort is performed. If `topn` is specified and no maximum or minimum rule length is specified, then the only columns allowed in the sort order are `RULE_CONFIDENCE` and `RULE_SUPPORT`. If `topn` is specified and a maximum or minimum rule length *is* specified, then `RULE_CONFIDENCE`, `RULE_SUPPORT`, and `NUMBER_OF_ITEMS` are allowed in the sort order.

Table 42-79 (Cont.) GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
`rule_id`	Identifier of the rule to return. If you specify a value for `rule_id`, do not specify values for the other filtering parameters.
`min_confidence`	Returns the rules with confidence greater than or equal to this number.
`min_support`	Returns the rules with support greater than or equal to this number.
`max_rule_length`	Returns the rules with a length less than or equal to this number. Rule length refers to the number of items in the rule (See `NUMBER_OF_ITEMS` in Table 42-80). For example, in the rule A=>B (if A, then B), the number of items is 2. If `max_rule_length` is specified, then the `NUMBER_OF_ITEMS` column is permitted in the sort order.
`min_rule_length`	Returns the rules with a length greater than or equal to this number. See `max_rule_length` for a description of rule length. If `min_rule_length` is specified, then the `NUMBER_OF_ITEMS` column is permitted in the sort order.
`sort_order`	Sorts the rules by the values in one or more of the returned columns. Specify one or more column names, each followed by `ASC` for ascending order or `DESC` for descending order. (See Table 42-80 for the column names.) For example, to sort the result set in descending order first by the `NUMBER_OF_ITEMS` column, then by the `RULE_CONFIDENCE` column, you must specify: `ORA_MINING_VARCHAR2_NT('NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC')` If you specify `topn`, the results will vary depending on the sort order. By default, the results are sorted by Confidence in descending order, then by Support in descending order.
`antecedent_items`	Returns the rules with these items in the antecedent.
`consequent_items`	Returns the rules with this item in the consequent.
`min_lift`	Returns the rules with lift greater than or equal to this number.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

The object type returned by `GET_ASSOCIATION_RULES` is described in Table 42-80. For descriptions of each field, see the Usage Notes.

Table 42-80 GET_ASSOCIATION RULES Function Return Values

Return Value	Description

DM_RULES	A set of rows of type DM_RULE. The rows have the following columns:
	`(rule_id INTEGER,`
	` antecedent DM_PREDICATES,`
	` consequent DM_PREDICATES,`
	` rule_support NUMBER,`
	` rule_confidence NUMBER,`
	` rule_lift NUMBER,`
	` antecedent_support NUMBER,`
	` consequent_support NUMBER,`
	` number_of_items INTEGER )`

DM_PREDICATES	The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/,`
	` attribute_num_value NUMBER,`
	` attribute_str_value VARCHAR2(4000),`
	` attribute_support NUMBER,`
	` attribute_confidence NUMBER)`

Usage Notes

1. This table function pipes out rows of type DM_RULES. For information on machine learning data types and piped output from table functions, see "Datatypes".
2. The columns returned by GET_ASSOCIATION_RULES are described as follows:

Column in DM_RULES	Description
rule_id	Unique identifier of the rule

Column in DM_RULES	Description

antecedent	The independent condition in the rule. When this condition exists, the dependent condition in the consequent also exists.
	The condition is a combination of attribute values called a predicate (DM_PREDICATE). The predicate specifies a condition for each attribute. The condition may specify equality (=), inequality (<>), greater than (>), less than (<), greater than or equal to (>=), or less than or equal to (<=) a given value.
	Support and Confidence for each attribute condition in the antecedent is returned in the predicate. Support is the number of transactions that satisfy the antecedent. Confidence is the likelihood that a transaction will satisfy the antecedent.
	Note: The occurrence of the attribute as a DM_PREDICATE indicates the presence of the item in the transaction. The actual value for attribute_num_value or attribute_str_value is meaningless. For example, the following predicate indicates that 'Mouse Pad' is present in the transaction *even though* the attribute value is NULL.
	`DM_PREDICATE('PROD_NAME',`
	`'Mouse Pad', '= ', NULL, NULL, NULL,`
	`NULL))`

consequent	The dependent condition in the rule. This condition exists when the antecedent exists. The consequent, like the antecedent, is a predicate (DM_PREDICATE). Support and confidence for each attribute condition in the consequent is returned in the predicate. Support is the number of transactions that satisfy the consequent. Confidence is the likelihood that a transaction will satisfy the consequent.
rule_support	The number of transactions that satisfy the rule.
rule_confidence	The likelihood of a transaction satisfying the rule.
rule_lift	The degree of improvement in the prediction over random chance when the rule is satisfied.
antecedent_support	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions.
consequent_support	The ratio of the number of transactions that satisfy the consequent to the total number of transactions.
number_of_items	The total number of attributes referenced in the antecedent and consequent of the rule.

Examples

The following example demonstrates an association model build followed by several invocations of the GET_ASSOCIATION_RULES table function:

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
```

```
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

-- build an AR model
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'market_settings');
END;
/
-- View the (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model'));
```

In the previous example, you view all rules. To view just the top 20 rules, use the following statement.

```
-- View the top 20 (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model', 20));
```

The following query uses the association model `AR_SH_SAMPLE`.

```
SELECT * FROM TABLE (
   DBMS_DATA_MINING.GET_ASSOCIATION_RULES (
      'AR_SH_SAMPLE', 10, NULL, 0.5, 0.01, 2, 1,
         ORA_MINING_VARCHAR2_NT (
         'NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC', 'RULE_SUPPORT DESC'),
         DM_ITEMS(DM_ITEM('CUSTPRODS', 'Mouse Pad', 1, NULL),
                  DM_ITEM('CUSTPRODS', 'Standard Mouse', 1, NULL)),
         DM_ITEMS(DM_ITEM('CUSTPRODS', 'Extension Cable', 1, NULL))));
```

The query returns three rules, shown as follows:

```
13  DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL),
       DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
    .15532      .84393   2.7075     .18404     .3117   2

11  DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
    .18085      .56291   1.8059     .32128     .3117   1

9   DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
      .17766    .55116   1.7682     .32234     .3117   1
```

> **See Also:**
>
> Table 42-80 for the DM_RULE column data types.

42.1.8.21 GET_FREQUENT_ITEMSETS Function

The GET_FREQUENT_ITEMSETS function returns a set of rows that represent the frequent itemsets from an association model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead..

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

For a detailed description of frequent itemsets, consult *Oracle Machine Learning for SQL Concepts*.

Syntax

```
DBMS_DATA_MINING.get_frequent_itemsets(
      model_name IN VARCHAR2,
      topn IN NUMBER DEFAULT NULL,
      max_itemset_length IN NUMBER DEFAULT NULL,
      partition_name     IN VARCHAR2 DEFAULT NULL)
  RETURN DM_ItemSets PIPELINED;
```

Parameters

Table 42-81 GET_FREQUENT_ITEMSETS Function Parameters

Parameter	Description
model_name	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
topn	When not NULL, return the top *n* rows ordered by support in descending order
max_itemset_length	Maximum length of an item set.
partition_name	Specifies a partition in a partitioned model.

> **Note:**
>
> The partition_name columns applies only when the model is partitioned.

Return Values

Table 42-82 GET_FREQUENT_ITEMSETS Function Return Values

Return Value	Description

DM_ITEMSETS	A set of rows of type DM_ITEMSET. The rows have the following columns:
	`(partition_name VARCHAR2(128)`
	`itemsets_id NUMBER,`
	`items DM_ITEMS,`
	`support NUMBER,`
	`number_of_items NUMBER)`
	Note: The partition_name columns applies only when the model is partitioned.
	The items column returns a nested table of type DM_ITEMS. The rows have type DM_ITEM:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000))`

Usage Notes

This table function pipes out rows of type DM_ITEMSETS. For information on machine learning data types and piped output from table functions, see "Data Types".

Examples

The following example demonstrates an association model build followed by an invocation of GET_FREQUENT_ITEMSETS table function from Oracle SQL.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS

    SELECT *

  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

/* build a AR model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name           => 'market_model',
  function             => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name      => 'market_build',
  case_id_column_name  => 'item_id',
```

```
  target_column_name   => NULL,
  settings_table_name  => 'market_settings');
END;
/

-- View the (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model'));
```

In the example above, you view all itemsets. To view just the top 20 itemsets, use the following statement:

```
-- View the top 20 (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model', 20));
```

42.1.8.22 GET_MODEL_COST_MATRIX Function

The `GET_*` interfaces are replaced by model views, and Oracle recommends that users leverage the views instead.

The GET_MODEL_COST_MATRIX function is replaced by the `DM$VC` prefixed view, Scoring Cost Matrix. The cost matrix used when building a Decision Tree is made available by the `DM$VM` prefixed view, Decision Tree build cost matrix.

Refer to Model Detail View for Classification Algorithm.

The `GET_MODEL_COST_MATRIX` function returns the rows of a cost matrix associated with the specified model.

By default, this function returns the scoring cost matrix that was added to the model with the `ADD_COST_MATRIX` procedure. If you wish to obtain the cost matrix used to create a model, specify `cost_matrix_type_create` as the `matrix_type`. See Table 42-83.

See also ADD_COST_MATRIX Procedure.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_COST_MATRIX (
      model_name                IN VARCHAR2,
      matrix_type               IN VARCHAR2 DEFAULT cost_matrix_type_score)
      partition_name            IN VARCHAR2 DEFAULT NULL);
RETURN DM_COST_MATRIX PIPELINED;
```

Parameters

Table 42-83   GET_MODEL_COST_MATRIX Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`matrix_type`	The type of cost matrix. `COST_MATRIX_TYPE_SCORE` — cost matrix used for scoring. (Default.) `COST_MATRIX_TYPE_CREATE` — cost matrix used to create the model (Decision Tree only).
`partition_name`	Name of the partition in a partitioned model

Return Values

Table 42-84 GET_MODEL_COST_MATRIX Function Return Values

Return Value	Description

`DM_COST_MATRIX`	A set of rows of type `DM_COST_ELEMENT`. The rows have the following columns:
	`actual VARCHAR2(4000), NUMBER,`
	`predicted VARCHAR2(4000), cost`
	`NUMBER)`

Usage Notes

Only Decision Tree models can be built with a cost matrix. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See Table 42-7.

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, you can use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one.

The `GET_MODEL_COST_MATRIX` may return either the build or scoring cost matrix defined for a model or model partition.

If you do not specify a partitioned model name, then an error is displayed.

Example

This example returns the scoring cost matrix associated with the Naive Bayes model `NB_SH_CLAS_SAMPLE`.

```
column actual format a10
column predicted format a10
SELECT *
    FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
    ORDER BY predicted, actual;

ACTUAL     PREDICTED   COST
---------- ---------- -----
0          0            .00
1          0            .75
0          1            .25
1          1            .00
```

42.1.8.23 GET_MODEL_DETAILS_AI Function

The `GET_MODEL_DETAILS_AI` function returns a set of rows that provide the details of an attribute importance model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User’s Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_ai(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN dm_ranked_attributes pipelined;
```

Parameters

Table 42-85 GET_MODEL_DETAILS_AI Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 42-86 GET_MODEL_DETAILS_AI Function Return Values

Return Value	Description

`DM_RANKED_ATTRIBUTES`	A set of rows of type `DM_RANKED_ATTRIBUTE`. The rows have the following columns:
	`(attribute_name        VARCHAR2(4000,`
	` attribute_subname     VARCHAR2(4000),`
	` importance_value      NUMBER,`
	` rank                  NUMBER(38))`

Examples

The following example returns model details for the attribute importance model `AI_SH_sample`, which was created by the sample program `dmaidemo.sql`.

```
SELECT attribute_name, importance_value, rank
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_AI('AI_SH_sample'))
    ORDER BY RANK;

ATTRIBUTE_NAME                            IMPORTANCE_VALUE       RANK
---------------------------------------- ---------------- ----------
HOUSEHOLD_SIZE                                  .151685183          1
CUST_MARITAL_STATUS                             .145294546          2
YRS_RESIDENCE                                    .07838928          3
AGE                                             .075027496          4
Y_BOX_GAMES                                     .063039952          5
EDUCATION                                       .059605314          6
HOME_THEATER_PACKAGE                            .056458722          7
OCCUPATION                                      .054652937          8
CUST_GENDER                                     .035264741          9
BOOKKEEPING_APPLICATION                         .019204751         10
PRINTER_SUPPLIES                                         0         11
OS_DOC_SET_KANJI                                -.00050013         12
FLAT_PANEL_MONITOR                              -.00509564         13
BULK_PACK_DISKETTES                             -.00540822         14
```

```
COUNTRY_NAME                                              -.01201116         15
CUST_INCOME_LEVEL                                         -.03951311         16
```

42.1.8.24 GET_MODEL_DETAILS_EM Function

The `GET_MODEL_DETAILS_EM` function returns a set of rows that provide statistics about the clusters produced by an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

By default, the EM algorithm groups components into high-level clusters, and `GET_MODEL_DETAILS_EM` returns only the high-level clusters with their hierarchies. Alternatively, you can configure EM model to disable the grouping of components into high-level clusters. In this case, `GET_MODEL_DETAILS_EM` returns the components themselves as clusters with their hierarchies. See Table 42-12.

Syntax

```
DBMS_DATA_MINING.get_model_details_em(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      attribute_subname  VARCHAR2 DEFAULT NULL,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 42-87 GET_MODEL_DETAILS_EM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise, the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned

Table 42-87 (Cont.) GET_MODEL_DETAILS_EM Function Parameters

Parameter	Description
`attribute_subname`	The name of a nested attribute. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the column and *`attribute_subname`* is the name of the nested attribute in that column. If the attribute is not nested, then `attribute_subname` is null.
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *`n`* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *`topn`*, then, up to *`n`* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. For information on Oracle Machine Learning for SQL data types and return values for Clustering algorithms piped output from table functions, see "Data Types".
2. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.
3. When cluster statistics are disabled (`EMCS_CLUSTER_STATISTICS` is set to `EMCS_CLUS_STATS_DISABLE`), `GET_MODEL_DETAILS_EM` does not return centroids, histograms, or rules. Only taxonomy (hierarchy) and cluster counts are returned.
4. When the `partition_name` is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

42.1.8.25 GET_MODEL_DETAILS_EM_COMP Function

he `GET_MODEL_DETAILS_EM_COMP` table function returns a set of rows that provide details about the parameters of an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_em_comp(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_EM_COMPONENT_SET PIPELINED;
```

Parameters

Table 42-88 GET_MODEL_DETAILS_EM_COMP Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model to retrieve details for.

Return Values

Table 42-89 GET_MODEL_DETAILS_EM_COMP Function Return Values

Return Value	Description

`DM_EM_COMPONENT_SET`	A set of rows of type `DM_EM_COMPONENT`. The rows have the following columns:
	`(info_type VARCHAR2(30),`
	`component_id NUMBER,`
	`cluster_id NUMBER,`
	`attribute_name VARCHAR2(4000),`
	`covariate_name VARCHAR2(4000),`
	`attribute_value VARCHAR2(4000),`
	`value NUMBER )`

Usage Notes

1. This table function pipes out rows of type `DM_EM_COMPONENT`. For information on Oracle Machine Learning for SQL data types and piped output from table functions, see "Data Types".

   The columns in each row returned by `GET_MODEL_DETAILS_EM_COMP` are described as follows:

Column in DM_EM_COMPONENT	Description
`info_type`	The type of information in the row. The following information types are supported: • `cluster` • `prior` • `mean` • `covariance` • `frequency`
`component_id`	Unique identifier of a component
`cluster_id`	Unique identifier of the high-level leaf cluster for each component

Column in DM_EM_COMPONENT	Description
attribute_name	Name of an original attribute or a derived feature ID. The derived feature ID is used in models built on data with nested columns. The derived feature definitions can be obtained from the GET_MODEL_DETAILS_EM_PROJ Function.
covariate_name	Name of an original attribute or a derived feature ID used in variance/covariance definition
attribute_value	Categorical value or bin interval for binned numerical attributes
value	Encodes different information depending on the value of info_type, as follows: • cluster — The value field is NULL • prior — The value field returns the component prior • mean — The value field returns the mean of the attribute specified in attribute_name • covariance — The value field returns the covariance of the attributes specified in attribute_name and covariate_name. Using the same attribute in attribute_name and covariate_name, returns the variance. • frequency— The value field returns the multivalued Bernoulli frequency parameter for the attribute/value combination specified by attribute_name and attribute_value See Usage Note 2 for details.

2. The following table shows which fields are used for each info_type. The blank cells represent NULLs.

info_type	component_id	cluster_id	attribute_name	covariate_name	attribute_value	value
cluster	X	X				
prior	X	X				X
mean	X	X	X			X
covariance	X	X	X	X		X
frequency	X	X	X		X	X

3. GET_MODEL_DETAILS functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.
4. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

42.1.8.26 GET_MODEL_DETAILS_EM_PROJ Function

The `GET_MODEL_DETAILS_EM_PROJ` function returns a set of rows that provide statistics about the projections produced by an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_em_proj(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_EM_PROJECTION_SET PIPELINED;
```

Parameters

Table 42-90 GET_MODEL_DETAILS_EM_PROJ Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 42-91 GET_MODEL_DETAILS_EM_PROJ Function Return Values

Return Value	Description

`DM_EM_PROJECTION_SET`	A set of rows of type `DM_EM_PROJECTION`. The rows have the following columns:
	`(feature_name VARCHAR2(4000),`
	` attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` attribute_value VARCHAR2(4000),`
	` coefficient NUMBER )`
	See Usage Notes for details.

Usage Notes

1. This table function pipes out rows of type `DM_EM_PROJECTION`. For information on machine learning data types and piped output from table functions, see "Datatypes".

   The columns in each row returned by `GET_MODEL_DETAILS_EM_PROJ` are described as follows:

Column in DM_EM_PROJECTION	Description
`feature_name`	Name of a derived feature. The feature maps to the attribute_name returned by the GET_MODEL_DETAILS_EM Function.
`attribute_name`	Name of a column in the build data
`attribute_subname`	Subname in a nested column
`attribute_value`	Categorical value
`coefficient`	Projection coefficient. The representation is sparse; only the non-zero coefficients are returned.

2. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

   The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information.
3. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.27 GET_MODEL_DETAILS_GLM Function

The `GET_MODEL_DETAILS_GLM` function returns the coefficient statistics for a generalized linear model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

The same set of statistics is returned for both linear and logistic regression, but statistics that do not apply to the machine learning function are returned as `NULL`. For more details, see the Usage Notes.

Syntax

```
DBMS_DATA_MINING.get_model_details_glm(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_GLM_Coeff_Set PIPELINED;
```

Parameters

Table 42-92 GET_MODEL_DETAILS_GLM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 42-93 GET_MODEL_DETAILS_GLM Return Values

Return Value	Description

`DM_GLM_COEFF_SET`	A set of rows of type `DM_GLM_COEFF`. The rows have the following columns:
	`(class VARCHAR2(4000),`
	`attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_value VARCHAR2(4000),`
	`feature_expression VARCHAR2(4000),`
	`coefficient NUMBER,`
	`std_error NUMBER,`
	`test_statistic NUMBER,`
	`p_value NUMBER,`
	`VIF NUMBER,`
	`std_coefficient NUMBER,`
	`lower_coeff_limit NUMBER,`
	`upper_coeff_limit NUMBER,`
	`exp_coefficient BINARY_DOUBLE,`
	`exp_lower_coeff_limit BINARY_DOUBLE,`
	`exp_upper_coeff_limit BINARY_DOUBLE)`

`GET_MODEL_DETAILS_GLM` returns a row of statistics for each attribute and one extra row for the intercept, which is identified by a null value in the attribute name. Each row has the `DM_GLM_COEFF` data type. The statistics are described in Table 42-94.

Table 42-94 DM_GLM_COEFF Data Type Description

Column	Description
`class`	The non-reference target class for logistic regression. The model is built to predict the probability of this class. The other class (the reference class) is specified in the model setting `GLMS_REFERENCE_CLASS_NAME`. See Table 42-19. For Linear Regression, `class` is null.
`attribute_name`	The attribute name when there is no subname, or first part of the attribute name when there is a subname. The value of `attribute_name` is also the name of the column in the case table that is the source for this attribute. For the intercept, `attribute_name` is null. Intercepts are equivalent to the bias term in SVM models.
`attribute_subname`	The name of an attribute in a nested table. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the nested column in the case table that is the source for this attribute. If the attribute is not nested, then `attribute_subname` is null. If the attribute is an intercept, then both the `attribute_name` and the `attribute_subname` are null.

Table 42-94 (Cont.) DM_GLM_COEFF Data Type Description

Column	Description
`attribute_value`	The value of the attribute (categorical attribute only). For numeric attributes, `attribute_value` is null.
`feature_expression`	The feature name constructed by the algorithm when feature generation is enabled and higher-order features are found. If feature selection is not enabled, then the feature name is simply the fully-qualified attribute name (*`attribute_name.attribute_subname`* if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form: *`fully-qualified_attribute_name.attribute_value`* For numeric attributes, the algorithm constructs a name for the higher-order feature by taking the product of the resulting values: (*`attrib1`*)*(*`attrib2`*))*...... where *`attrib1`* and *`attrib2`* are fully-qualified attribute names.
`coefficient`	The linear coefficient estimate.
`std_error`	Standard error of the coefficient estimate.
`test_statistic`	For linear regression, the t-value of the coefficient estimate. For logistic regression, the Wald chi-square value of the coefficient estimate.
`p-value`	Probability of the `test_statistic`. Used to analyze the significance of specific attributes in the model.
`VIF`	Variance Inflation Factor. The value is zero for the intercept. For logistic regression, `VIF` is null. VIF is not computed if the solver is Cholesky.
`std_coefficient`	Standardized estimate of the coefficient.
`lower_coeff_limit`	Lower confidence bound of the coefficient.
`upper_coeff_limit`	Upper confidence bound of the coefficient.
`exp_coefficient`	Exponentiated coefficient for logistic regression. For linear regression, `exp_coefficient` is null.
`exp_lower_coeff_limit`	Exponentiated coefficient for lower confidence bound of the coefficient for logistic regression. For linear regression, `exp_lower_coeff_limit` is null.
`exp_upper_coeff_limit`	Exponentiated coefficient for upper confidence bound of the coefficient for logistic regression. For linear regression, `exp_lower_coeff_limit` is null.

Usage Notes

Not all statistics are necessarily returned for each coefficient. Statistics will be null if:

- They do not apply to the machine learning function. For example, `exp_coefficient` does not apply to linear regression.
- They cannot be computed from a theoretical standpoint. For information on ridge regression, see Table 42-19.

- They cannot be computed because of limitations in system resources.
- Their values would be infinity.
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns some of the model details for the GLM regression model `GLMR_SH_Regr_sample`.

```
SET line 120
SET pages 99
column attribute_name format a30
column attribute_subname format a20
column attribute_value format a20
col coefficient format 990.9999
col std_error format 990.9999
SQL> SELECT * FROM
(SELECT attribute_name, attribute_value, coefficient, std_error
  FROM DM$VDGLMR_SH_REGR_SAMPLE order by 1,2)
WHERE rownum < 11;

ATTRIBUTE_NAME                 ATTRIBUTE_VALUE      COEFFICIENT     STD_ERROR
------------------------------ -------------------- ----------- ---------
AFFINITY_CARD                                          -0.5797         0.5283
BOOKKEEPING_APPLICATION                              -0.4689         3.8872
BULK_PACK_DISKETTES                                  -0.9819         2.5430
COUNTRY_NAME                     Argentina              -1.2020         1.1876
COUNTRY_NAME                     Australia              -0.0071         5.1146
COUNTRY_NAME                     Brazil                  5.2931         1.9233
COUNTRY_NAME                     Canada                   4.0191         2.4108
COUNTRY_NAME                     China                   0.8706         3.5889
COUNTRY_NAME                     Denmark              -2.9822         3.1803
COUNTRY_NAME                     France                   -1.1044         7.1811
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.28 GET_MODEL_DETAILS_GLOBAL Function

The `GET_MODEL_DETAILS_GLOBAL` function returns statistics about the model as a whole. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Global details are available for Generalized Linear Models, Association Rules, Singular Value Decomposition, and Expectation Maximization. There are new Global model views which show global information for all algorithms. Oracle recommends that users leverage the views instead. Refer to Model Details View Global.

Syntax

```
DBMS_DATA_MINING.get_model_details_global(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_model_global_details PIPELINED;
```

Parameters

Table 42-95 GET_MODEL_DETAILS_GLOBAL Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 42-96 GET_MODEL_DETAILS_GLOBAL Function Return Values

Return Value	Description

`DM_MODEL_GLOBAL_DETAILS`	A collection of rows of type `DM_MODEL_GLOBAL_DETAIL`. The rows have the following columns:
	`(global_detail_name   VARCHAR2(30),`
	` global_detail_value   NUMBER)`

Examples

The following example returns the global model details for the GLM regression model `GLMR_SH_Regr_sample`.

```
SELECT *
  FROM TABLE(dbms_data_mining.get_model_details_global(
              'GLMR_SH_Regr_sample'))
ORDER BY global_detail_name;
GLOBAL_DETAIL_NAME             GLOBAL_DETAIL_VALUE
------------------------------ -------------------
ADJUSTED_R_SQUARE                       .731412557
AIC                                       5931.814
COEFF_VAR                               18.1711243
CORRECTED_TOTAL_DF                            1499
CORRECTED_TOT_SS                        278740.504
DEPENDENT_MEAN                              38.892
ERROR_DF                                      1433
ERROR_MEAN_SQUARE                       49.9440956
ERROR_SUM_SQUARES                       71569.8891
F_VALUE                                 62.8492452
GMSEP                                    52.280819
HOCKING_SP                              .034877162
J_P                                     52.1749319
MODEL_CONVERGED                                  1
MODEL_DF                                        66
MODEL_F_P_VALUE                                  0
MODEL_MEAN_SQUARE                       3138.94871
MODEL_SUM_SQUARES                       207170.615
NUM_PARAMS                                      67
NUM_ROWS                                      1500
ROOT_MEAN_SQ                            7.06711367
R_SQ                                    .743238288
SBIC                                    6287.79977
VALID_COVARIANCE_MATRIX                          1
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.29 GET_MODEL_DETAILS_KM Function

The `GET_MODEL_DETAILS_KM` function returns a set of rows that provide the details of a *k*-means clustering model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

You can provide input to `GET_MODEL_DETAILS_KM` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, then `GET_MODEL_DETAILS_KM` returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.get_model_details_km(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      attribute_subname  VARCHAR2 DEFAULT NULL,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 42-97 GET_MODEL_DETAILS_KM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned

Table 42-97 (Cont.) GET_MODEL_DETAILS_KM Function Parameters

Parameter	Description
`attribute_subname`	The name of a nested attribute. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the column and *`attribute_subname`* is the name of the nested attribute in that column. If the attribute is not nested, `attribute_subname` is null.
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *n* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *topn*, then up to *n* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. The table function pipes out rows of type `DM_CLUSTERS`. For information on machine learning data types and Return Value for Clustering Algorithms piped output from table functions, see "Data Types".
2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the *k*-means clustering model `KM_SH_Clus_sample`.

```
SELECT T.id           clu_id,
       T.record_count rec_cnt,
       T.parent       parent,
       T.tree_level   tree_level,
       T.dispersion   dispersion
  FROM (SELECT *
          FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_KM(
                     'KM_SH_Clus_sample'))
        ORDER BY id) T
 WHERE ROWNUM < 6;

    CLU_ID    REC_CNT     PARENT TREE_LEVEL DISPERSION
---------- ---------- ---------- ---------- ----------
         1       1500                     1  5.9152211
         2        638          1          2 3.98458982
         3        862          1          2 5.83732097
         4        376          3          3 5.05192137
         5        486          3          3 5.42901522
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.30 GET_MODEL_DETAILS_NB Function

The `GET_MODEL_DETAILS_NB` function returns a set of rows that provide the details of a naive Bayes model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_nb(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_NB_Details PIPELINED;
```

Parameters

Table 42-98 GET_MODEL_DETAILS_NB Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 42-99 GET_MODEL_DETAILS_NB Function Return Values

Return Value	Description

`DM_NB_DETAILS`	A set of rows of type `DM_NB_DETAIL`. The rows have the following columns:
	`(target_attribute_name VARCHAR2(30),`
	`target_attribute_str_value VARCHAR2(4000),`
	`target_attribute_num_value NUMBER,`
	`prior_probability NUMBER,`
	`conditionals DM_CONDITIONALS)`
	The `conditionals` column of `DM_NB_DETAIL` returns a nested table of type `DM_CONDITIONALS`. The rows, of type `DM_CONDITIONAL`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_num_value NUMBER,`
	`conditional_probability NUMBER)`

Usage Notes

- The table function pipes out rows of type `DM_NB_DETAILS`. For information on machine learning data types and piped output from table functions, see "Data Types".

- When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following query is from the sample program `dmnbdemo.sql`. It returns model details about the model `NB_SH_Clas_sample`. For information about the sample programs, see *Oracle Machine Learning for SQL User's Guide*.

The query creates labels from the bin boundary tables that were used to bin the training data. It replaces the attribute values with the labels. For numeric bins, the labels are `(`*`lower_boundary`*`,`*`upper_boundary`*`]`; for categorical bins, the label matches the value it represents. (This method of categorical label representation will only work for cases where one value corresponds to one bin.) The target was not binned.

```
WITH
    bin_label_view AS (
    SELECT col, bin, (DECODE(bin,'1','[','(') || lv || ',' || val || ']') label
      FROM (SELECT col,
                   bin,
                   LAST_VALUE(val) OVER (
                   PARTITION BY col ORDER BY val
                   ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) lv,
                   val
             FROM nb_sh_sample_num)
   UNION ALL
   SELECT col, bin, val label
     FROM nb_sh_sample_cat
   ),
   model_details AS (
   SELECT T.target_attribute_name                                        tname,
          NVL(TO_CHAR(T.target_attribute_num_value,T.target_attribute_str_value)) tval,
          C.attribute_name                                               pname,
          NVL(L.label, NVL(C.attribute_str_value, C.attribute_num_value)) pval,
          T.prior_probability                                           priorp,
          C.conditional_probability                                      condp
     FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NB('NB_SH_Clas_sample')) T,
          TABLE(T.conditionals) C,
          bin_label_view L
    WHERE C.attribute_name = L.col (+) AND
          (NVL(C.attribute_str_value,C.attribute_num_value) = L.bin(+))
   ORDER BY 1,2,3,4,5,6
   )
   SELECT tname, tval, pname, pval, priorp, condp
     FROM model_details
    WHERE ROWNUM < 11;

TNAME          TVAL PNAME                     PVAL          PRIORP   CONDP
-------------- ---- ------------------------- ------------- -------  -------
AFFINITY_CARD  0    AGE                       (24,30]         .6500   .1714
AFFINITY_CARD  0    AGE                       (30,35]         .6500   .1509
AFFINITY_CARD  0    AGE                       (35,40]         .6500   .1125
AFFINITY_CARD  0    AGE                       (40,46]         .6500   .1134
AFFINITY_CARD  0    AGE                       (46,53]         .6500   .1071
AFFINITY_CARD  0    AGE                       (53,90]         .6500   .1312
AFFINITY_CARD  0    AGE                       [17,24]         .6500   .2134
AFFINITY_CARD  0    BOOKKEEPING_APPLICATION   0               .6500   .1500
AFFINITY_CARD  0    BOOKKEEPING_APPLICATION   1               .6500   .8500
AFFINITY_CARD  0    BULK_PACK_DISKETTES       0               .6500   .3670
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.31 GET_MODEL_DETAILS_NMF Function

The `GET_MODEL_DETAILS_NMF` function returns a set of rows that provide the details of a non-negative matrix factorization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_nmf(
      model_name IN VARCHAR2,
      partition_name VARCHAR2 DEFAULT NULL)
   RETURN DM_NMF_Feature_Set PIPELINED;
```

Parameters

Table 42-100 GET_MODEL_DETAILS_NMF Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 42-101 GET_MODEL_DETAILS_NMF Function Return Values

Return Value	Description

`DM_NMF_FEATURE_SET`	A set of rows of `DM_NMF_FEATURE`. The rows have the following columns:
	`(feature_id NUMBER,`
	` mapped_feature_id VARCHAR2(4000),`
	` attribute_set DM_NMF_ATTRIBUTE_SET)`
	The `attribute_set` column of `DM_NMF_FEATURE` returns a nested table of type `DM_NMF_ATTRIBUTE_SET`. The rows, of type `DM_NMF_ATTRIBUTE`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` attribute_value VARCHAR2(4000),`
	` coefficient NUMBER)`

Usage Notes

- The table function pipes out rows of type `DM_NMF_FEATURE_SET`. For information on machine learning data types and piped output from table functions, see "Data Types".

- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the feature extraction model `NMF_SH_Sample`.

```
SELECT * FROM (
SELECT F.feature_id,
       A.attribute_name,
       A.attribute_value,
       A.coefficient
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF('NMF_SH_Sample')) F,
       TABLE(F.attribute_set) A
ORDER BY feature_id,attribute_name,attribute_value
) WHERE ROWNUM < 11;

FEATURE_ID ATTRIBUTE_NAME          ATTRIBUTE_VALUE          COEFFICIENT
--------- ----------------------- ---------------- -------------------
        1 AFFINITY_CARD                              .051208078859308
        1 AGE                                       .0390513260041573
        1 BOOKKEEPING_APPLICATION                   .0512734004239326
        1 BULK_PACK_DISKETTES                        .232471260895683
        1 COUNTRY_NAME            Argentina        .00766817464479959
        1 COUNTRY_NAME            Australia       .000157637881096675
        1 COUNTRY_NAME            Brazil            .0031409632415604
        1 COUNTRY_NAME            Canada           .00144213099311427
        1 COUNTRY_NAME            China           .000102279310968754
        1 COUNTRY_NAME            Denmark         .000242424084307513
```

Related Topics

- *Oracle Machine Learning for SQL User’s Guide*

42.1.8.32 GET_MODEL_DETAILS_OC Function

The `GET_MODEL_DETAILS_OC` function returns a set of rows that provide the details of an O-cluster clustering model. The rows are an enumeration of the clustering patterns generated during the creation of the model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User’s Guide*.

You can provide input to `GET_MODEL_DETAILS_OC` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, then `GET_MODEL_DETAILS_OC` returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.get_model_details_oc(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 42-102 GET_MODEL_DETAILS_OC Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *n* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *topn*, then up to *n* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. For information about machine learning data types and return values for clustering algorithms piped output from table functions, see "Data Types".
2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the clustering model `OC_SH_Clus_sample`.

For each cluster in this example, the split predicate indicates the attribute and the condition used to assign records to the cluster's children during model build. It provides an important piece of information on how the population within a cluster can be divided up into two smaller clusters.

```
SELECT clu_id, attribute_name, op, s_value
     FROM (SELECT a.id clu_id, sp.attribute_name, sp.conditional_operator op,
```

```
                    sp.attribute_str_value s_value
               FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_OC(
                      'OC_SH_Clus_sample')) a,
                    TABLE(a.split_predicate) sp
             ORDER BY a.id, op, s_value)
       WHERE ROWNUM < 11;

     CLU_ID ATTRIBUTE_NAME       OP S_VALUE
----------- -------------------- ---------------------------------
          1 OCCUPATION           IN ?
          1 OCCUPATION           IN Armed-F
          1 OCCUPATION           IN Cleric.
          1 OCCUPATION           IN Crafts
          2 OCCUPATION           IN ?
          2 OCCUPATION           IN Armed-F
          2 OCCUPATION           IN Cleric.
          3 OCCUPATION           IN Exec.
          3 OCCUPATION           IN Farming
          3 OCCUPATION           IN Handler
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.33 GET_MODEL_SETTINGS Function

The `GET_MODEL_SETTINGS` function returns the settings used to build the given model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

Syntax

```
FUNCTION get_model_settings(model_name IN VARCHAR2)
  RETURN DM_Model_Settings PIPELINED;
```

Parameters

Table 42-103 GET_MODEL_SETTINGS Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Return Values

Table 42-104 GET_MODEL_SETTINGS Function Return Values

Return Value	Description

`DM_MODEL_SETTINGS`	A set of rows of type `DM_MODEL_SETTINGS`. The rows have the following columns:
	`DM_MODEL_SETTINGS TABLE OF SYS.DM_MODEL_SETTING`
	`Name Type`
	`---------------------- --------------------`
	`SETTING_NAME VARCHAR2(30)`
	`SETTING_VALUE VARCHAR2(4000)`

Usage Notes

1. This table function pipes out rows of type `DM_MODEL_SETTINGS`. For information on machine learning data types and piped output from table functions, see "DBMS_DATA_MINING Datatypes".
2. The setting names/values include both those specified by the user and any defaults assigned by the build process.

Examples

The following example returns model model settings for an example naive Bayes model.

```
SETTING_NAME                   SETTING_VALUE
------------------------------ ------------------------------
ALGO_NAME                       ALGO_NAIVE_BAYES
PREP_AUTO                       ON
ODMS_MAX_PARTITIONS             1000
NABS_SINGLETON_THRESHOLD       0
CLAS_WEIGHTS_BALANCED          OFF
NABS_PAIRWISE_THRESHOLD        0
ODMS_PARTITION_COLUMNS         GENDER,Y_BOX_GAMES
ODMS_MISSING_VALUE_TREATMENT   ODMS_MISSING_VALUE_AUTO
ODMS_SAMPLING                  ODMS_SAMPLING_DISABLE

9 rows selected.
```

Related Topics

- *Oracle Database Reference*

42.1.8.34 GET_MODEL_SIGNATURE Function

The `GET_MODEL_SIGNATURE` function returns the list of columns from the build input table that were used by the build process to train the model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

Syntax

```
FUNCTION get_model_signature (model_name IN VARCHAR2)
RETURN DM_Model_Signature PIPELINED;
```

Parameters

Table 42-105 GET_MODEL_SIGNATURE Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Return Values

Table 42-106 GET_MODEL_SIGNATURE Function Return Values

Return Value	Description

`DM_MODEL_SIGNATURE`	A set of rows of type `DM_MODEL_SIGNATURE`. The rows have the following columns:
	`DM_MODEL_SIGNATURE TABLE OF SYS.DM_MODEL_SIGNATURE_ATTRIBUTE`
	`Name                Type`
	`------------------  -------------------`
	`ATTRIBUTE_NAME      VARCHAR2(130)`
	`ATTRIBUTE_TYPE      VARCHAR2(106)`

Usage Notes

1. This table function pipes out rows of type `DM_MODEL_SIGNATURE`. For information on machine learning data types and piped output from table functions, see "DBMS_DATA_MINING Datatypes".
2. The signature names or types include only those attributes used by the build process.

Examples

The following example returns model settings for an example naive Bayes model.

```
ATTRIBUTE_NAME                 ATTRIBUTE_TYPE
------------------------------ ------------------
AGE                            NUMBER
ANNUAL_INCOME                  NUMBER
AVERAGE___ITEMS_PURCHASED      NUMBER
BOOKKEEPING_APPLICATION        NUMBER
BULK_PACK_DISKETTES            NUMBER
BULK_PURCH_AVE_AMT             NUMBER
DISABLE_COOKIES                NUMBER
EDUCATION                      VARCHAR2
FLAT_PANEL_MONITOR             NUMBER
GENDER                         VARCHAR2
HOME_THEATER_PACKAGE           NUMBER
HOUSEHOLD_SIZE                 VARCHAR2
MAILING_LIST                   NUMBER
MARITAL_STATUS                 VARCHAR2
NO_DIFFERENT_KIND_ITEMS        NUMBER
OCCUPATION                     VARCHAR2
OS_DOC_SET_KANJI               NUMBER
PETS                           NUMBER
PRINTER_SUPPLIES               NUMBER
PROMO_RESPOND                  NUMBER
SHIPPING_ADDRESS_COUNTRY       VARCHAR2
SR_CITIZEN                     NUMBER
TOP_REASON_FOR_SHOPPING        VARCHAR2
WKS_SINCE_LAST_PURCH           NUMBER
WORKCLASS                      VARCHAR2
YRS_RESIDENCE                  NUMBER
Y_BOX_GAMES                    NUMBER

27 rows selected.
```

Related Topics

- *Oracle Database Reference*

42.1.8.35 GET_MODEL_DETAILS_SVD Function

The `GET_MODEL_DETAILS_SVD` function returns a set of rows that provide the details of a singular value decomposition model. Oracle recommends to use model details view settings. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

Refer to Model Details View for Singular Value Decomposition.

Syntax

```
DBMS_DATA_MINING.get_model_details_svd(
      model_name IN VARCHAR2,
      matrix_type IN VARCHAR2 DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
   RETURN DM_SVD_MATRIX_Set PIPELINED;
```

Parameters

Table 42-107 GET_MODEL_DETAILS_SVD Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`matrix_type`	Specifies which of the three SVD matrix types to return. Values are: `U`, `S`, `V`, and `NULL`. When `matrix_type` is null (default), all matrices are returned. The U matrix is only computed when the `SVDS_U_MATRIX_OUTPUT` setting is enabled. It is not computed by default. If the model does not contain U matrices and you set `matrix_type` to `U`, an empty set of rows is returned. See Table 42-27.
`partition_name`	A partition in a partitioned model.

Return Values

Table 42-108 GET_MODEL_DETAILS_SVD Function Return Values

Return Value	Description

`DM_SVD_MATRIX_SET`	A set of rows of type `DM_SVD_MATRIX`. The rows have the following columns:
	`(matrix_type CHAR(1),`
	`feature_id NUMBER,`
	`mapped_feature_id VARCHAR2(4000),`
	`attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`case_id VARCHAR2(4000),`
	`value NUMBER,`
	`variance NUMBER,`
	`pct_cum_variance NUMBER)`
	See Usage Notes for details.

Usage Notes

1. This table function pipes out rows of type `DM_SVD_MATRIX`. For information on machine learning data types and piped output from table functions, see "Data Types".

   The columns in each row returned by `GET_MODEL_DETAILS_SVD` are described as follows:

Column in DM_SVD_MATRIX_SET	Description
`matrix_type`	The type of matrix. Possible values are **S**, **V**, and **U**. This field is never null.
`feature_id`	The feature that the matrix entry refers to.
`mapped_feature_id`	A descriptive name for the feature.
`attribute_name`	Column name in the **V** matrix component bases. This field is null for the **S** and **U** matrices.
`attribute_subname`	Subname in the **V** matrix component bases. This is relevant only in the case of a nested column. This field is null for the **S** and **U** matrices.
`case_id`	Unique identifier of the row in the build data described by the **U** matrix projection. This field is null for the **S** and **V** matrices.
`value`	The matrix entry value.
`variance`	The variance explained by a component. It is non-null only for **S** matrix entries. This column is non-null only for **S** matrix entries and for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` and the build data is centered, either manually or because the setting `dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.

Column in DM_SVD_MATRIX_SET	Description
`pct_cum_variance`	The percent cumulative variance explained by the components thus far. The components are ranked by the explained variance in descending order. This column is non-null only for **S** matrix entries and for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` and the build data is centered, either manually or because the setting `dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.

2. The output of `GET_MODEL_DETAILS` is in sparse format. Zero values are not returned. Only the diagonal elements of the **S** matrix, the non-zero coefficients in the **V** matrix bases, and the non-zero **U** matrix projections are returned.

   There is one exception: If the data row does not produce non-zero **U** Matrix projections, the case ID for that row is returned with `NULL` for the `feature_id` and `value`. This is done to avoid losing any records from the original data.
3. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.
4. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the preferred partition name.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.36 GET_MODEL_DETAILS_SVM Function

The `GET_MODEL_DETAILS_SVM` function returns a set of rows that provide the details of a linear support vector machines (SVM) model. If invoked for nonlinear SVM, it returns `ORA-40215`. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

In linear SVM models, only nonzero coefficients are stored. This reduces storage and speeds up model loading. As a result, if an attribute is missing in the coefficient list returned by `GET_MODEL_DETAILS_SVM`, then the coefficient of this attribute should be interpreted as zero.

Syntax

```
DBMS_DATA_MINING.get_model_details_svm(
      model_name    VARCHAR2,
      reverse_coef NUMBER DEFAULT 0,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN DM_SVM_Linear_Coeff_Set PIPELINED;
```

Parameters

Table 42-109 GET_MODEL_DETAILS_SVM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`reverse_coef`	Whether or not `GET_MODEL_DETAILS_SVM` should transform the attribute coefficients using the original attribute transformations. When `reverse_coef` is set to 0 (default), `GET_MODEL_DETAILS_SVM` returns the coefficients directly from the model without applying transformations. When `reverse_coef` is set to 1, `GET_MODEL_DETAILS_SVM` transforms the coefficients and bias by applying the normalization shifts and scales that were generated using automatic data preparation. See Usage Note 4.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 42-110 GET_MODEL_DETAILS_SVM Function Return Values

Return Value	Description

`DM_SVM_LINEAR_COEFF_SET`	A set of rows of type `DM_SVM_LINEAR_COEFF`. The rows have the following columns:
	`(class            VARCHAR2(4000),`
	` attribute_set     DM_SVM_ATTRIBUTE_SET)`
	The `attribute_set` column returns a nested table of type `DM_SVM_ATTRIBUTE_SET`. The rows, of type `DM_SVM_ATTRIBUTE`, have the following columns:
	`(attribute_name        VARCHAR2(4000),`
	` attribute_subname   VARCHAR2(4000),`
	` attribute_value      VARCHAR2(4000),`
	` coefficient           NUMBER)`
	See Usage Notes.

Usage Notes

1. This table function pipes out rows of type `DM_SVM_LINEAR_COEFF`. For information on machine learning data types and piped output from table functions, see "Data Types".
2. The `class` column of `DM_SVM_LINEAR_COEFF` contains classification target values. For SVM Regression models, `class` is null. For each classification target value, a set of coefficients is returned. For binary classification, one-class classification, and regression models, only a single set of coefficients is returned.
3. The `attribute_value` column in `DM_SVM_ATTRIBUTE_SET` is used for categorical attributes.
4. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the

model details are the original attributes (or a close approximation of the original attributes) used to build the model.

The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information. If you want `GET_MODEL_DETAILS_SVM` to transform the coefficients such that they relate to the original attributes, set the `reverse_coef` parameter to 1.

5. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the SVM classification model `SVMC_SH_Clas_sample`, which was created by the sample program `dmsvcdem.sql`. For information about the sample programs, see *Oracle Machine Learning for SQL User's Guide*.

```
WITH
  mod_dtls AS (
  SELECT *
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('SVMC_SH_Clas_sample'))
  ),
  model_details AS (
  SELECT D.class, A.attribute_name, A.attribute_value, A.coefficient
    FROM mod_dtls D,
          TABLE(D.attribute_set) A
    ORDER BY D.class, ABS(A.coefficient) DESC
  )
  SELECT class, attribute_name aname, attribute_value aval, coefficient coeff
    FROM model_details
    WHERE ROWNUM < 11;

CLASS      ANAME                     AVAL                      COEFF
---------- ------------------------- ------------------------- -----
1                                                              -2.85
1          BOOKKEEPING_APPLICATION                              1.11
1          OCCUPATION                Other                      -.94
1          HOUSEHOLD_SIZE            4-5                         .88
1          CUST_MARITAL_STATUS       Married                     .82
1          YRS_RESIDENCE                                         .76
1          HOUSEHOLD_SIZE            6-8                        -.74
1          OCCUPATION                Exec.                       .71
1          EDUCATION                 11th                       -.71
1          EDUCATION                 Masters                     .63
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

42.1.8.37 GET_MODEL_DETAILS_XML Function

This function returns an XML object that provides the details of a decision tree model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views for Decision Tree in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_xml(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN XMLType;
```

Parameters

Table 42-111 GET_MODEL_DETAILS_XML Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 42-112 GET_MODEL_DETAILS_XML Function Return Value

Return Value	Description
`XMLTYPE`	The XML definition for the decision tree model. See "XMLTYPE" for details. The XML definition conforms to the Data Mining Group Predictive Model Markup Language (PMML) version 2.1 specification. The specification is available at `https://dmg.org`. If a nested attribute is used as a splitter, the attribute will appear in the XML document as field="'<column_name>'.<subname>", as opposed to the non-nested attributes which appear in the document as field="<column_name>". **Note:** The column names are surrounded by single quotes and a period separates the column_name from the subname. The rest of the document style remains unchanged.

Usage Notes

Special characters that cannot be displayed by Oracle XML are converted to '#'.

Examples

The following statements in SQL*Plus return the details of the decision tree model `dt_sh_clas_sample`.

Note: The "&quot" characters you will see in the XML output are a result of SQL*Plus behavior. To display the XML in proper format, cut and past it into a file and open the file in a browser.

```
column dt_details format a320
SELECT
 dbms_data_mining.get_model_details_xml('dt_sh_clas_sample')
 AS DT_DETAILS
FROM dual;


DT_DETAILS
--------------------------------------------------------------------------------
<PMML version="2.1">
  <Header copyright="Copyright (c) 2004, Oracle Corporation. All rights
      reserved."/>
  <DataDictionary numberOfFields="9">
    <DataField name="AFFINITY_CARD" optype="categorical"/>
    <DataField name="AGE" optype="continuous"/>
    <DataField name="BOOKKEEPING_APPLICATION" optype="continuous"/>
    <DataField name="CUST_MARITAL_STATUS" optype="categorical"/>
    <DataField name="EDUCATION" optype="categorical"/>
    <DataField name="HOUSEHOLD_SIZE" optype="categorical"/>
    <DataField name="OCCUPATION" optype="categorical"/>
    <DataField name="YRS_RESIDENCE" optype="continuous"/>
    <DataField name="Y_BOX_GAMES" optype="continuous"/>
  </DataDictionary>
  <TreeModel modelName="DT_SH_CLAS_SAMPLE" functionName="classification"
      splitCharacteristic="binarySplit">
    <Extension name="buildSettings">
      <Setting name="TREE_IMPURITY_METRIC" value="TREE_IMPURITY_GINI"/>
      <Setting name="TREE_TERM_MAX_DEPTH" value="7"/>
      <Setting name="TREE_TERM_MINPCT_NODE" value=".05"/>
      <Setting name="TREE_TERM_MINPCT_SPLIT" value=".1"/>
      <Setting name="TREE_TERM_MINREC_NODE" value="10"/>
      <Setting name="TREE_TERM_MINREC_SPLIT" value="20"/>
      <costMatrix>
        <costElement>
          <actualValue>0</actualValue>
          <predictedValue>0</predictedValue>
          <cost>0</cost>
        </costElement>
        <costElement>
          <actualValue>0</actualValue>
          <predictedValue>1</predictedValue>
          <cost>1</cost>
        </costElement>
        <costElement>
          <actualValue>1</actualValue>
          <predictedValue>0</predictedValue>
          <cost>8</cost>
        </costElement>
        <costElement>
          <actualValue>1</actualValue>
          <predictedValue>1</predictedValue>
          <cost>0</cost>
        </costElement>
      </costMatrix>
    </Extension>
    <MiningSchema>
      .
      .
      .
      .
      .
```

```
        .
      </Node>
    </Node>
  </TreeModel>
</PMML>
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

42.1.8.38 GET_MODEL_TRANSFORMATIONS Function

This function returns the transformation expressions embedded in the specified model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

All `GET_*` interfaces are replaced by model views, and Oracle recommends that users reference the model views to retrieve the relevant information. The `GET_MODEL_TRANSFORMATIONS` function is replaced by the following:

- USER(/DBA/ALL)_MINING_MODEL_XFORMS: provides the user-embedded transformations
- DM$VX prefixed model view: provides text feature extraction information
- D$VN prefixed mode view: provides normalization and missing value information
- DM$VB: provides binning information

> **See Also:**
>
> "About Transformation Lists" in DBMS_DATA_MINING_TRANSFORM Operational Notes
>
> "GET_TRANSFORM_LIST Procedure"
>
> "CREATE_MODEL Procedure"
>
> "ALL_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> "DBA_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> "USER_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> Model Details View for Binning
>
> Normalization and Missing Value Handling
>
> Data Preparation for Text Features

Syntax

```
DBMS_DATA_MINING.get_model_transformations(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_Transforms PIPELINED;
```

Parameters

Table 42-113 GET_MODEL_TRANSFORMATIONS Function Parameters

Parameter	Description
`model_name`	Indicates the name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 42-114 GET_MODEL_TRANSFORMATIONS Function Return Value

Return Value	Description

`DM_TRANSFORMS`	The transformation expressions embedded in *`model_name`*.
	The `DM_TRANSFORMS` type is a table of `DM_TRANSFORM` objects. Each `DM_TRANSFORM` has these fields:
	`attribute_name        VARCHAR2(4000)`
	`attribute_subname     VARCHAR2(4000)`
	`expression            CLOB`
	`reverse_expression    CLOB`

Usage Notes

When Automatic Data Preparation (ADP) is enabled, both automatic and user-defined transformations may be associated with an attribute. In this case, the user-defined transformations are evaluated before the automatic transformations.

When invoked for a partitioned model, the `partition_name` parameter must be specified.

Examples

In this example, several columns in the `SH.CUSTOMERS` table are used to create a naive Bayes model. A transformation expression is specified for one of the columns. The model does not use ADP.

```
CREATE OR REPLACE VIEW mining_data AS
   SELECT cust_id, cust_year_of_birth, cust_income_level,cust_credit_limit
   FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_INCOME_LEVEL                                  VARCHAR2(30)
 CUST_CREDIT_LIMIT                                  NUMBER

CREATE TABLE settings_nb(
      setting_name  VARCHAR2(30),
      setting_value VARCHAR2(30));
```

```
BEGIN
     INSERT INTO settings_nb (setting_name, setting_value) VALUES
           (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
     INSERT INTO settings_nb (setting_name, setting_value) VALUES
           (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_off);
     COMMIT;
END;
/
DECLARE
    mining_data_xforms   dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.SET_TRANSFORM (
         xform_list           =>  mining_data_xforms,
         attribute_name       => 'cust_year_of_birth',
         attribute_subname    =>  null,
         expression           => 'cust_year_of_birth + 10',
         reverse_expression   => 'cust_year_of_birth - 10');
    dbms_data_mining.CREATE_MODEL (
        model_name           =>  'new_model',
        mining_function      =>   dbms_data_mining.classification,
        data_table_name      =>  'mining_data',
        case_id_column_name  =>  'cust_id',
        target_column_name   =>  'cust_income_level',
        settings_table_name  =>  'settings_nb',
        data_schema_name     =>   nulL,
        settings_schema_name =>   null,
        xform_list           =>   mining_data_xforms );
  END;
 /
SELECT attribute_name, TO_CHAR(expression), TO_CHAR(reverse_expression)
      FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('new_model'));

ATTRIBUTE_NAME      TO_CHAR(EXPRESSION)       TO_CHAR(REVERSE_EXPRESSION)
------------------  ------------------------  ----------------------------
CUST_YEAR_OF_BIRTH  cust_year_of_birth + 10   cust_year_of_birth - 10
```

Related Topics

- *Oracle Database Reference*

42.1.8.39 GET_TRANSFORM_LIST Procedure

This procedure converts transformation expressions specified as `DM_TRANSFORMS` to a transformation list (`TRANSFORM_LIST`) that can be used in creating a model. `DM_TRANSFORMS` is returned by the `GET_MODEL_TRANSFORMATIONS` function.

You can also use routines in the `DBMS_DATA_MINING_TRANSFORM` package to construct a transformation list.

> **See Also:**
>
> "About Transformation Lists" in DBMS_DATA_MINING_TRANSFORM
>
> "GET_MODEL_TRANSFORMATIONS Function"
>
> "CREATE_MODEL Procedure"

Syntax

```
DBMS_DATA_MINING.GET_TRANSFORM_LIST (
      xform_list           OUT NOCOPY TRANSFORM_LIST,
      model_xforms         IN   DM_TRANSFORMS);
```

Parameters

Table 42-115 GET_TRANSFORM_LIST Procedure Parameters

Parameter	Description

`xform_list`	A list of transformation specifications that can be embedded in a model. Accepted as a parameter to the CREATE_MODEL Procedure.
	The `TRANSFORM_LIST` type is a table of `TRANSFORM_REC` objects. Each `TRANSFORM_REC` has these fields:
	`attribute_name        VARCHAR2(30)`
	`attribute_subname     VARCHAR2(4000)`
	`expression            EXPRESSION_REC`
	`reverse_expression    EXPRESSION_REC`
	`attribute_spec        VARCHAR2(4000)`
	For details about the `TRANSFORM_LIST` collection type, see Table 42-123.

`model_xforms`	A list of embedded transformation expressions returned by the GET_MODEL_TRANSFORMATIONS Function for a specific model.
	The `DM_TRANSFORMS` type is a table of `DM_TRANSFORM` objects. Each `DM_TRANSFORM` has these fields:
	`attribute_name         VARCHAR2(4000)`
	`attribute_subname      VARCHAR2(4000)`
	`expression             CLOB`
	`reverse_expression     CLOB`

Examples

In this example, a model `mod1` is trained using several columns in the `SH.CUSTOMERS` table. The model uses ADP, which automatically bins one of the columns.

A second model `mod2` is trained on the same data without ADP, but it uses a transformation list that was obtained from `mod1`. As a result, both `mod1` and `mod2` have the same embedded transformation expression.

```
CREATE OR REPLACE VIEW mining_data AS
     SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit
     FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_INCOME_LEVEL                                  VARCHAR2(30)
 CUST_CREDIT_LIMIT                                  NUMBER

CREATE TABLE setmod1(setting_name  VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
```

```
   INSERT INTO setmod1 VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
   INSERT INTO setmod1 VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
   dbms_data_mining.CREATE_MODEL (
               model_name            => 'mod1',
               mining_function       => dbms_data_mining.classification,
               data_table_name       => 'mining_data',
               case_id_column_name   => 'cust_id',
               target_column_name    => 'cust_income_level',
               settings_table_name   => 'setmod1');
    COMMIT;
END;
/
CREATE TABLE setmod2(setting_name  VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
  INSERT INTO setmod2
      VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
  COMMIT;
END;
/
DECLARE
  v_xform_list       dbms_data_mining_transform.TRANSFORM_LIST;
  dmxf               DM_TRANSFORMS;
BEGIN
   EXECUTE IMMEDIATE
    'SELECT dm_transform(attribute_name, attribute_subname,expression, reverse_expression)
     FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS (''mod1''))'
     BULK COLLECT INTO dmxf;
   dbms_data_mining.GET_TRANSFORM_LIST (
        xform_list             =>  v_xform_list,
        model_xforms           =>  dmxf);
   dbms_data_mining.CREATE_MODEL(
         model_name            => 'mod2',
         mining_function       =>  dbms_data_mining.classification,
         data_table_name       => 'mining_data',
         case_id_column_name   => 'cust_id',
         target_column_name    => 'cust_income_level',
         settings_table_name   => 'setmod2',
         xform_list            =>  v_xform_list);
END;
/

-- Transformation expression embedded in mod1
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(EXPRESSION)
--------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Transformation expression embedded in mod2
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(EXPRESSION)
--------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
```

```
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Reverse transformation expression embedded in mod1
SELECT TO_CHAR(reverse_expression)FROM TABLE
(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5;  )',NULL,'NULL')

-- Reverse transformation expression embedded in mod2
SELECT TO_CHAR(reverse_expression) FROM TABLE
(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5;  )',NULL,'NULL')
```

42.1.8.40 IMPORT_MODEL Procedure

This procedure imports one or more machine learning models. The procedure is overloaded. You can call it to import machine learning models from a dump file set, or you can call it to import a single machine learning model from a PMML document.

Import from a dump file set

You can import machine learning models from a dump file set that was created by the EXPORT_MODEL Procedure. `IMPORT_MODEL` and `EXPORT_MODEL` use Oracle Data Pump technology to export to and import from a dump file set.

When Oracle Data Pump is used directly to export/import an entire schema or database, the machine learning models in the schema or database are included. `EXPORT_MODEL` and `IMPORT_MODEL` export/import machine learning models only.

Import from PMML

You can import a machine learning model represented in Predictive Model Markup Language (PMML). The model must be of type `RegressionModel`, either linear regression or binary logistic regression.

PMML is an XML-based standard specified by the Data Mining Group (`https://dmg.org`). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Machine Learning for SQL supports the core features of PMML 3.1 for regression models.

See Also:

Oracle Machine Learning for SQL User's Guide for more information about exporting and importing machine learning models

Oracle Database Utilities for information about Oracle Data Pump

`https://dmg.org/dmg-faq.html` for more information about PMML

Syntax

Imports a machine learning model from a dump file set:

```
DBMS_DATA_MINING.IMPORT_MODEL (
      filename          IN  VARCHAR2,
      directory         IN  VARCHAR2,
      model_filter      IN  VARCHAR2 DEFAULT NULL,
      operation         IN  VARCHAR2 DEFAULT NULL,
      remote_link       IN  VARCHAR2 DEFAULT NULL,
      jobname           IN  VARCHAR2 DEFAULT NULL,
      schema_remap      IN  VARCHAR2 DEFAULT NULL,
      tablespace_remap  IN  VARCHAR2 DEFAULT NULL);
```

Imports a machine learning model from a PMML document:

```
DBMS_DATA_MINING.IMPORT_MODEL (
      model_name        IN  VARCHAR2,
      pmmldoc           IN  XMLTYPE
      strict_check      IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 42-116 IMPORT_MODEL Procedure Parameters

Parameter	Description
`filename`	Name of the dump file set from which the models should be imported. The dump file set must have been created by the `EXPORT_MODEL` procedure or the `expdp` export utility of Oracle Data Pump. The dump file set can contain one or more files. (Refer to "EXPORT_MODEL Procedure" for details.) If the dump file set contains multiple files, you can specify '*filename*%U' instead of listing them. For example, if your dump file set contains 3 files, `archive01.dmp`, `archive02.dmp`, and `archive03.dmp`, you can import them by specifying `'archive%U'`.
`directory`	Name of a pre-defined directory object that specifies where the dump file set is located. Both the exporting and the importing user must have read/write access to the directory object and to the file system directory that it identifies. Note: The target database must have also have read/write access to the file system directory.

Table 42-116 (Cont.) IMPORT_MODEL Procedure Parameters

Parameter	Description

`model_filter`	Optional parameter that specifies one or more models to import. If you do not specify a value for `model_filter`, all models in the dump file set are imported. You can also specify `NULL` (the default) or 'ALL' to import all models.
	The value of `model_filter` can be one or more model names. The following are valid filters.
	`'mymodel1'`
	`'name IN (''mymodel2'',''mymodel3'')'`
	The first causes `IMPORT_MODEL` to import a single model named `mymodel1`. The second causes `IMPORT_MODEL` to import two models, `mymodel2` and `mymodel3`.

`operation`	Optional parameter that specifies whether to import the models or the SQL statements that create the models. By default, the models are imported. You can specify either of the following values for `operation`: • `'IMPORT'` — Import the models (Default) • `'SQL_FILE'`— Write the SQL DDL for creating the models to a text file. The text file is named *`job_name`*`.sql` and is located in the dump set directory.
`remote_link`	Optional parameter that specifies the name of a database link to a remote system. The default value is `NULL`. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for `remote_link`, you can import models into the local database from the remote database. The import is fileless; no dump file is involved. The `IMP_FULL_DATABASE` role is required for importing the remote models. The `EXP_FULL_DATABASE` privilege, the `CREATE DATABASE LINK` privilege, and other privileges may also be required. (See Example 2.)
`jobname`	Optional parameter that specifies the name of the import job. By default, the name has the form *`username`*`_imp_`*`nnnn`*, where *`nnnn`* is a number. For example, a job name in the `SCOTT` schema might be `SCOTT_imp_134`. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the import job, named *`jobname`*`.log,` is created in the same directory as the dump file set.
`schema_remap`	Optional parameter for importing into a different schema. By default, models are exported and imported within the same schema. If the dump file set belongs to a different schema, you must specify a schema mapping in the form *`export_user`*:*`import_user`*. For example, you would specify `'SCOTT:MARY'` to import a model exported by `SCOTT` into the `MARY` schema. Note: In some cases, you may need to have the `IMP_FULL_DATABASE` privilege or the `SYS` role to import a model from a different schema.
`tablespace_remap`	Optional parameter for importing into a different tablespace. By default, models are exported and imported within the same tablespace. If the dump file set belongs to a different tablespace, you must specify a tablespace mapping in the form *`export_tablespace`*:*`import_tablespace`*. For example, you would specify `'TBLSPC01:TBLSPC02'` to import a model that was exported from tablespace `TBLSPC01` into tablespace `TBLSPC02`. Note: In some cases, you may need to have the `IMP_FULL_DATABASE` privilege or the `SYS` role to import a model from a different tablespace.

Table 42-116 (Cont.) IMPORT_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name for the new model that will be created in the database as a result of an import from PMML The name must be unique within the user's schema.
`pmmldoc`	The PMML document representing the model to be imported. The PMML document has an `XMLTYPE` object type. See "XMLTYPE" for details.
`strict_check`	Whether or not an error occurs when the PMML document contains sections that are not part of core PMML (for example, Output or Targets). OML4SQL supports only core PMML; any non-core features may affect the scoring representation. If the PMML does not strictly conform to core PMML and `strict_check` is set to `TRUE`, then `IMPORT_MODEL` returns an error. If strict_check is `FALSE` (the default), then the error is suppressed. The model may be imported and scored.

Examples

1. This example shows a model being exported and imported within the schema `oml_user2`. Then the same model is imported into the `oml_user3` schema. The `oml_user3` user has the `IMP_FULL_DATABASE` privilege. The `oml_user2` user has been assigned the `USER2` tablespace; `oml_user3` has been assigned the `USER3` tablespace.

```
SQL> connect oml_user2
Enter password: oml_user2_password
Connected.
SQL> select model_name from user_mining_models;

MODEL_NAME
------------------------------
NMF_SH_SAMPLE
SVMO_SH_CLAS_SAMPLE
SVMR_SH_REGR_SAMPLE

-- export the model called NMF_SH_SAMPLE to a dump file in same schema
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL (
            filename =>'NMF_SH_SAMPLE_out',
            directory =>'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''');

-- import the model back into the same schema
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (
            filename => 'NMF_SH_SAMPLE_out01.dmp',
            directory => 'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''');

-- connect as different user
-- import same model into that schema
SQL> connect oml_user3
Enter password: oml_user3_password
Connected.
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (
            filename => 'NMF_SH_SAMPLE_out01.dmp',
            directory => 'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''',
            operation =>'IMPORT',
            remote_link => NULL,
```

```
            jobname => 'nmf_imp_job',
            schema_remap => 'oml_user2:oml_user3',
            tablespace_remap => 'USER2:USER3');
```

The following example shows user `MARY` importing all models from a dump file, `model_exp_001.dmp`, which was created by user `SCOTT`. User `MARY` has been assigned a tablespace named `USER2`; user `SCOTT` was assigned the tablespace `USERS` when the models were exported into the dump file `model_exp_001.dmp`.The dump file is located in the file system directory mapped to a directory object called `DM_DUMP`. If user `MARY` does not have `IMP_FULL_DATABASE` privileges, `IMPORT_MODEL` will raise an error.

```
-- import all models
DECLARE
  file_name  VARCHAR2(40);
BEGIN
  file_name := 'model_exp_001.dmp';
  DBMS_DATA_MINING.IMPORT_MODEL(
            filename=> 'file_name',
            directory=>'DM_DUMP',
            schema_remap=>'SCOTT:MARY',
            tablespace_remap=>'USERS:USER2');
  DBMS_OUTPUT.PUT_LINE(
           'DBMS_DATA_MINING.IMPORT_MODEL of all models from SCOTT done!');
END;
/
```

2. This example shows how the user `xuser` could import the model `oml_user.r1mod` from a remote database. The SQL*Net connection alias for the remote database is `R1DB`. The user `xuser` is assigned the `SYSAUX` tablespace; the user `oml_user` is assigned the `TBS_1` tablespace.

```
CONNECT / AS SYSDBA;
GRANT CREATE DATABASE LINK TO xuser;
GRANT imp_full_database TO xuser;
CONNECT xuser/xuserpassword
CREATE DATABASE LINK oml_user_link
         CONNECT TO oml_user IDENTIFIED BY oml_userpassword USING 'R1DB';
EXEC dbms_data_mining.import_model (
    NULL,
   'oml_user_DIR',
   'R1MOD',
    remote_link => 'oml_user_LINK', schema_remap => 'oml_user:XUSER',
                    tablespace_remap => 'TBS_1:SYSAUX' );
SELECT name FROM dm_user_models;

NAME
-----------------------------------------------------------------------------
R1MOD
```

3. This example shows how a PMML document called `SamplePMML1.xml` could be imported from a location referenced by directory object `PMMLDIR` into the schema of the current user. The imported model will be called `PMMLMODEL1`.

```
BEGIN
    dbms_data_mining.import_model ('PMMLMODEL1',
        XMLType (bfilename ('PMMLDIR', 'SamplePMML1.xml'),
          nls_charset_id ('AL32UTF8')
        ));
END;
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

42.1.8.41 IMPORT_SERMODEL Procedure

This procedure imports the serialized format of the model back into a database.

The import routine takes the serialized content in the `BLOB` and the name of the model to be created with the content. This import does not create model views or tables that are needed for querying model details. The import procedure only provides the ability to score the model.

Syntax

```
DBMS_DATA_MINING.IMPORT_SERMODEL (
      model_data     IN BLOB,
      model_name     IN VARCHAR2,);
```

Parameters

Table 42-117   IMPORT_SERMODEL Procedure Parameters

Parameter	Description
`model_data`	Provides model data in `BLOB` format.
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Examples

The following statement imports the serialized format of the models.

```
declare
 v_blob blob;
BEGIN
 dbms_lob.createtemporary(v_blob, FALSE);
-- fill in v_blob from somewhere (e.g., bfile, etc.)
 dbms_data_mining.import_sermodel(v_blob, 'MY_MODEL');
 dbms_lob.freetemporary(v_blob);
END;
/
```

Related Topics

- EXPORT_SERMODEL Procedure
  This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

See Also:

Oracle Machine Learning for SQL User's Guide for more information about exporting and importing machine learning models

42.1.8.42 IMPORT_ONNX_MODEL Procedure

This procedure enables you to import an ONNX model into the Database.

Syntax

```
DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
model_name   IN  VARCHAR2,
model_data   IN  BLOB,
metadata     IN  JSON);
```

Parameters

Table 42-118 IMPORT_ONNX_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form `[schema_name.]model_name`. If you do not specify a schema, then your own schema is used.
`model_data`	It is a `BLOB` holding the ONNX representation of the model. The `BLOB` contains the identical byte sequence as the one stored in an ONNX file.
`metadata`	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models .

Example

The following example illustrates a code snippet of using the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure. The complete step-by-step example is illustrated in Import ONNX Models and Generate Embeddings and Alternate Method to Import ONNX Models.

```
DBMS_DATA_MINING.IMPORT_ONNX_MODEL('my_embedding_model.onnx',
                                             :blob_bind_variable,
                                              JSON('{"function" :
"embedding",

"embeddingOutput" : "embedding" ,
                                                      "input":
{"input": ["DATA"]}}'));
```

For a complete example to illustrate how you can define a `BLOB` variable and use it in the `IMPORT_ONNX_MODEL` procedure, you can have the following:

```
CREATE OR REPLACE MY_LOAD_EMBEDDING_MODEL(embedding_model_name
VARCHAR2, onnx_blob BLOB) IS
```

```
BEGIN
DBMS_DATA_MINING.IMPORT_ONNX_MODEL(embedding_model_name,
                            onnx_blob,
                            JSON('{"function" : "embedding",
                                   "embeddingOutput" : "embedding" ,
                                   "input":{"input": ["DATA"]}}'));
END;
/
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.
- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.
- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.

42.1.8.43 JSON Schema for R Extensible Algorithm

Provides some flexibility when creating a new JSON object following the JSON schema.

Usage Note

Some flexibility when creating a new JSON object is as follows:

- Partial registration is allowed. For example, the detail function can be missing.
- Different orders are allowed. For example, the detail function can be written before the build function or after it.

Example 42-1 JSON Schema

JSON schema 1.1 for R extensible algorithm:

```
{
    "type": "object",
    "properties": {
        "algo_name_display": { "type" : "object",
                                             "properties" : {
                                             "language" : { "type" :
"string",

"enum" : ["English", "Spanish", "French"],

"default" : "English"},
                                             "name" : { "type" : "string"}}
                                           },

        "function_language": {"type": "string" },
        "mining_function": {
```

```
                  "type" : "array",
                  "items" : [
                      { "type" : "object",
                         "properties" : {
                            "mining_function_name"  : { "type" :
"string"},
                            "build_function": {
                                    "type": "object",
                                    "properties": {
                                         "function_body": { "type":
"CLOB" }
                                                         }
                                     },

         "detail_function": {
                  "type" : "array",
                   "items" : [
                       {"type": "object",
                         "properties": {
                              "function_body": { "type": "CLOB" },
                              "view_columns": { "type" : "array",

"items" : {

    "type" : "object",

    "properties" : {

        "name" : { "type" : "string"},

        "type" : { "type" : "string",

                        "enum" : ["VARCHAR2",

                                         "NUMBER",

                                         "DATE",

                                         "BOOLEAN"]

                       }

     }
                                                           }
                                             }
                                  }
                      ]
         },

        "score_function": {
                  "type": "object",
                  "properties": {
                        "function_body": { "type": "CLOB" }
                         }
                  },
```

```
          "weight_function": {
                          "type": "object",
                          "properties": {
                              "function_body": { "type": "CLOB" },
                          }
                   }
                                 }
             }]
          },

         "algo_setting": {
                 "type" : "array",
                 "items" : [
                     { "type" : "object",
                        "properties" : {
                           "name"               : { "type" : "string"},
                           "name_display": { "type" : "object",
                                                            "properties" : {
                                                            "language" :
{ "type" : "string",

    "enum" : ["English", "Spanish", "French"],

    "default" : "English"},
                                                            "name" : { "type" :
"string"}}
                                                         },
                           "type" : { "type" : "string",
                                           "enum" : ["string", "integer",
"number", "boolean"]},

                           "optional": {"type" : "BOOLEAN",
                                                "default" : "FALSE"},

                           "value" : { "type" :  "string"},

                           "min_value" : { "type": "object",
                                                         "properties": {
                                                               "min_value":
{"type": "number"},
                                                                "inclusive":
{ "type": "boolean",

      "default" : TRUE},
                                                          }
                                                     },
                            "max_value" : {"type": "object",
                                                        "properties": {
                                                             "max_value":
{"type": "number"},
                                                             "inclusive":
{ "type": "boolean",

    "default" : TRUE},
                                                                  }
```

```
                                                    },
                          "categorical choices" : { "type": "array",
"items": {
"type": "string"
                                                               }
                                                           },
                          "description_display": { "type" : "object",
"properties" : {
"language" : { "type" : "string",
                   "enum" : ["English", "Spanish", "French"],
                   "default" : "English"},
"name" : { "type" : "string"}}
                                                           }
                          }
                      }
                   ]
            }
     }
}
```

Example 42-2   JSON object example

The following is an JSON object example that must be passed to the registration procedure:

```
{  "algo_name_display"   :     {"English", "t1"},
                         "function_language"    :      "R",
                         "mining_function" : {
  "mining_function_name" : "CLASSIFICATION",
                         "build_function" : {"function_body":
"function(dat, formula, family)
{

set.seed(1234);
                                          mod <- glm(formula =
formula, data=dat,
                                                      family=
eval(parse(text=family)));
mod}"},
           "score_function" :  { "function_body": "function(mod, dat) {
                                             res <- predict(mod,
newdata = dat,
type=''response
                                                ''));
                                             res2=data.frame(1-res,
res); res2}"}}
```

```
                          },
                          "algo_setting" :   [{"name"                :
"dbms_data_mining.odms_m


                                                          issing_value_treatment",
                            "name_display"   : {"English",
"dbms_data_mining.odms_missing_value
_treatment"},
                            "type"                 : "string",
                            "optional"           :  "TRUE",
                            "value"                :
"dbms_data_mining.odms_missing_value_mean_mode",
                            "categorical choices"   :
[    "dbms_data_mining.odms_missing_value_mean_mode",

"dbms_data_mining.odms_missing_value_auto",

"dbms_data_mining.odms_missing_value_delete_row"],
                            "description"                : {"English",
                                                                        "how to
treat missing values"}
                         },

{"name"                : "RALG_PARAMETER_FAMILY",
                            "name_display"   : {"English",
"RALG_PARAMETER_FAMILY"},
                            "type"                 : "string",
                            "optional"           :  "TRUE",
                            "value"                :  "",
                            "description"       : {"English", "R family
parameter in build function"}
                         }
],
                        }
```


42.1.8.44 REGISTER_ALGORITHM Procedure

Use this function to register a new algorithm by providing the algorithm name, machine learning function, and all other algorithm metadata.

Syntax

```
DBMS_DATA_MINING.REGISTER_ALGORITHM (
                  algorithm_name            IN VARCHAR2,
                  algorithm_metadata        IN CLOB,
                  algorithm_description     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-119 ***REGISTER_ALGORITHM Procedure Parameters***

Parameter	Description
`algorithm_name`	Name of the algorithm.

Table 42-119 (Cont.) *REGISTER_ALGORITHM Procedure Parameters*

Parameter	Description
`algorithm_metadata`	Metadata of the algorithm.
`algorithm_description`	Description of the algorithm.

Usage Notes

The registration procedure performs the following:

- Checks whether `algorithm_metadata` has correct JSON syntax.
- Checks whether the input JSON object follows the predefined JSON schema.
- Checks whether current user has `RQADMIN` privilege.
- Checks duplicate algorithms so that the same algorithm is not registered twice.
- Checks for missing entries. For example, algorithm name, algorithm type, metadata, and build function.

Register Algorithms After the JSON Object Is Created

SQL users can register new algorithms by creating a JSON object following the JSON schema and passing it to the `REGISTER_ALGORITHM` procedure.

```
BEGIN
  DBMS_DATA_MINING.register_algorithm(
    algorithm_name                 =>   't1',
    algorithm_metadata            =>
    '{"function_language" : "R",
      "mining_function" :
        { "mining_function_name" : "CLASSIFICATION",
           "build_function" : {"function_body": "function(dat,
formula, family) { set.seed(1234);
                                          mod <- glm(formula =
formula, data=dat,

family=eval(parse(text=family)));
mod}"},
           "score_function" :  {"function_body": "function(mod, dat) {
                                             res <- predict(mod,
newdata = dat, type=''response'');
                                             res2=data.frame(1-res,
res); res2}"}}
    }',
    algorithm_description  => 't1');
END;
/
```

42.1.8.45 RANK_APPLY Procedure

This procedure ranks the results of an APPLY operation based on a top-N specification for predictive and descriptive model results.

For classification models, you can provide a cost matrix as input, and obtain the ranked results with costs applied to the predictions.

Syntax

```
DBMS_DATA_MINING.RANK_APPLY (
      apply_result_table_name        IN VARCHAR2,
      case_id_column_name            IN VARCHAR2,
      score_column_name              IN VARCHAR2,
      score_criterion_column_name    IN VARCHAR2,
      ranked_apply_table_name        IN VARCHAR2,
      top_N                          IN NUMBER (38) DEFAULT 1,
      cost_matrix_table_name         IN VARCHAR2    DEFAULT NULL,
      apply_result_schema_name       IN VARCHAR2    DEFAULT NULL,
      cost_matrix_schema_name        IN VARCHAR2    DEFAULT NULL);
```

Parameters

Table 42-120 RANK_APPLY Procedure Parameters

Parameter	Description
apply_result_table_name	Name of the table or view containing the results of an APPLY operation on the test data set (see Usage Notes)
case_id_column_name	Name of the case identifier column. This must be the same as the one used for generating APPLY results.
score_column_name	Name of the prediction column in the apply results table
score_criterion_column_name	Name of the probability column in the apply results table
ranked_apply_result_tab_name	Name of the table containing the ranked apply results
top_N	Top N predictions to be considered from the APPLY results for precision recall computation
cost_matrix_table_name	Name of the cost matrix table
apply_result_schema_name	Name of the schema hosting the APPLY results table
cost_matrix_schema_name	Name of the schema hosting the cost matrix table

Usage Notes

You can use RANK_APPLY to generate ranked apply results, based on a top-N filter and also with application of cost for predictions, if the model was built with costs.

The behavior of RANK_APPLY is similar to that of APPLY with respect to other DDL-like operations such as CREATE_MODEL, DROP_MODEL, and RENAME_MODEL. The procedure does not depend on the model; the only input of relevance is the apply results generated in a fixed schema table from APPLY.

The main intended use of RANK_APPLY is for the generation of the final APPLY results against the scoring data in a production setting. You can apply the model against test data using APPLY, compute various test metrics against various cost matrix tables, and use the candidate cost matrix for RANK_APPLY.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The case_id column will be the same case identifier column as that of the apply results.

Classification Models — NB and SVM

For numerical targets, the ranked results table will have the definition as shown:

```
(case_id        VARCHAR2/NUMBER,
prediction      NUMBER,
probability     NUMBER,
cost            NUMBER,
rank            INTEGER)
```

For categorical targets, the ranked results table will have the following definition:

```
(case_id        VARCHAR2/NUMBER,
prediction      VARCHAR2,
probability     NUMBER,
cost            NUMBER,
rank            INTEGER)
```

Clustering Using *k*-Means or O-Cluster

Clustering is an unsupervised machine learning function, and hence there are no targets. The results of an APPLY operation contains simply the cluster identifier corresponding to a case, and the associated probability. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the cluster ids ranked by top-N.

```
(case_id        VARCHAR2/NUMBER,
cluster_id      NUMBER,
probability     NUMBER,
rank            INTEGER)
```

Feature Extraction using NMF

Feature extraction is also an unsupervised machine learning function, and hence there are no targets. The results of an APPLY operation contains simply the feature identifier corresponding to a case, and the associated match quality. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the feature ids ranked by top-N.

```
(case_id         VARCHAR2/NUMBER,
feature_id       NUMBER,
match_quality    NUMBER,
rank             INTEGER)
```

Examples

```
BEGIN
/* build a model with name census_model.
 * (See example under CREATE_MODEL)
 */

/* if training data was pre-processed in any manner,
```

```
 * perform the same pre-processing steps on apply
 * data also.
 * (See examples in the section on DBMS_DATA_MINING_TRANSFORM)
 */

/* apply the model to data to be scored */
DBMS_DATA_MINING.RANK_APPLY(
  apply_result_table_name       => 'census_apply_result',
  case_id_column_name           => 'person_id',
  score_column_name             => 'prediction',
  score_criterion_column_name   => 'probability
  ranked_apply_result_tab_name  => 'census_ranked_apply_result',
  top_N                         => 3,
  cost_matrix_table_name        => 'census_cost_matrix');
END;
/

-- View Ranked Apply Results
SELECT *
  FROM census_ranked_apply_result;
```

42.1.8.46 REMOVE_COST_MATRIX Procedure

The REMOVE_COST_MATRIX procedure removes the default scoring matrix from a classification model.

> **See Also:**
>
> - "ADD_COST_MATRIX Procedure"
> - "REMOVE_COST_MATRIX Procedure"

Syntax

```
DBMS_DATA_MINING.REMOVE_COST_MATRIX (
      model_name    IN  VARCHAR2);
```

Parameters

Table 42-121 Remove_Cost_Matrix Procedure Parameters

Parameter	Description
model_name	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used.

Usage Notes

If the model is not in your schema, then REMOVE_COST_MATRIX requires the ALTER ANY MINING MODEL system privilege or the ALTER object privilege for the machine learning model.

Example

The naive Bayes model NB_SH_CLAS_SAMPLE has an associated cost matrix that can be used for scoring the model.

```
SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;

ACTUAL     PREDICTED        COST
---------- ---------- ----------
0          0                   0
1          0                 .75
0          1                 .25
1          1                   0
```

You can remove the cost matrix with `REMOVE_COST_MATRIX`.

```
SQL>EXECUTE dbms_data_mining.remove_cost_matrix('nb_sh_clas_sample');

SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;

no rows selected
```

42.1.8.47 RENAME_MODEL Procedure

This procedure changes the name of the machine learning model indicated by *model_name* to the name that you specify as *new_model_name*.

If a model with *new_model_name* already exists, then the procedure optionally renames *new_model_name* to *versioned_model_name* before renaming *model_name* to *new_model_name*.

The model name is in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used. For machine learning model naming restrictions, see the Usage Notes for "CREATE_MODEL Procedure".

Syntax

```
DBMS_DATA_MINING.RENAME_MODEL (
     model_name             IN VARCHAR2,
     new_model_name         IN VARCHAR2,
     versioned_model_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-122 RENAME_MODEL Procedure Parameters

Parameter	Description
`model_name`	Model to be renamed.
`new_model_name`	New name for the model *`model_name`*.
`versioned_model_name`	New name for the model *`new_model_name`* if it already exists.

Usage Notes

If you attempt to rename a model while it is being applied, then the model will be renamed but the apply operation will return indeterminate results.

Examples

1. This example changes the name of model census_model to census_model_2012.

```
BEGIN
  DBMS_DATA_MINING.RENAME_MODEL(
    model_name      => 'census_model',
    new_model_name  => 'census_model_2012');
END;
/
```

2. In this example, there are two classification models in the user's schema: clas_mod, the working model, and clas_mod_tst, a test model. The RENAME_MODEL procedure preserves clas_mod as clas_mod_old and makes the test model the new working model.

```
SELECT model_name FROM user_mining_models;
MODEL_NAME
-------------------------------------------------------------------
CLAS_MOD
CLAS_MOD_TST

BEGIN
  DBMS_DATA_MINING.RENAME_MODEL(
    model_name            => 'clas_mod_tst',
    new_model_name        => 'clas_mod',
    versioned_model_name  => 'clas_mod_old');
END;
/

SELECT model_name FROM user_mining_models;
MODEL_NAME
-------------------------------------------------------------------
CLAS_MOD
CLAS_MOD_OLD
```

42.2 DBMS_DATA_MINING_TRANSFORM

DBMS_DATA_MINING_TRANSFORM implements a set of transformations that are commonly used in machine learning.

This chapter contains the following topics:

- Overview
- Operational Notes
- Security Model
- Datatypes
- Constants
- Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

See Also:

- DBMS_DATA_MINING
- *Oracle Machine Learning for SQL User's Guide*

42.2.1 Using DBMS_DATA_MINING_TRANSFORM

This section contains topics that relate to using the `DBMS_DATA_MINING_TRANSFORM` package.

- Overview
- Operational Notes
- Security Model
- Datatypes
- Constants

42.2.1.1 DBMS_DATA_MINING_TRANSFORM Overview

A transformation is a SQL expression that modifies the data in one or more columns.

Data must typically undergo certain transformations before it can be used to build a machine learning model. Many machine learning algorithms have specific transformation requirements.

Data that will be scored must be transformed in the same way as the data that was used to create (train) the model.

External or Embedded Transformations

`DBMS_DATA_MINING_TRANSFORM` offers two approaches to implementing transformations. For a given model, you can either:

- Create a list of transformation expressions and pass it to the CREATE_MODEL Procedure

  or

- Create a view that implements the transformations and pass the name of the view to the CREATE_MODEL Procedure

If you create a transformation list and pass it to `CREATE_MODEL`, the transformation expressions are embedded in the model and automatically implemented whenever the model is applied.

If you create a view, the transformation expressions are external to the model. You will need to re-create the transformations whenever you apply the model.

Note:

Embedded transformations significantly enhance the model's usability while simplifying the process of model management.

Automatic Transformations

Oracle Machine Learning for SQL supports an Automatic Data Preparation (ADP) mode. When ADP is enabled, most algorithm-specific transformations are *automatically* embedded. Any additional transformations must be explicitly provided in an embedded transformation list or in a view.

If ADP is enabled and you create a model with a transformation list, both sets of transformations are embedded. The model will execute the user-specified transformations from the transformation list before executing the automatic transformations specified by ADP.

Within a transformation list, you can selectively disable ADP for individual attributes.

> **See Also:**
>
> "Automatic Data Preparation"
>
> *Oracle Machine Learning for SQL User's Guide* for a more information about ADP
>
> "DBMS_DATA_MINING_TRANSFORM-About Transformation Lists"

Transformations in DBMS_DATA_MINING_TRANSFORM

The transformations supported by `DBMS_DATA_MINING_TRANSFORM` are summarized in this section.

Binning

Binning refers to the mapping of continuous or discrete values to discrete values of reduced cardinality.

- Supervised Binning (Categorical and Numerical)

  Binning is based on intrinsic relationships in the data as determined by a decision tree model.

  See "INSERT_BIN_SUPER Procedure".
- Top-N Frequency Categorical Binning

  Binning is based on the number of cases in each category.

  See "INSERT_BIN_CAT_FREQ Procedure"
- Equi-Width Numerical Binning

  Binning is based on equal-range partitions.

  See "INSERT_BIN_NUM_EQWIDTH Procedure".
- Quantile Numerical Binning

  Binning is based on quantiles computed using the SQL `NTILE` function.

  See "INSERT_BIN_NUM_QTILE Procedure".

Linear Normalization

Normalization is the process of scaling continuous values down to a specific range, often between zero and one. Normalization transforms each numerical value by subtracting a number (the **shift**) and dividing the result by another number (the **scale**).

```
x_new = (x_old-shift)/scale
```

- Min-Max Normalization

  Normalization is based on the minimum and maximum with the following shift and scale:

  ```
  shift = min
  scale = max-min
  ```

  See "INSERT_NORM_LIN_MINMAX Procedure".
- Scale Normalization

  Normalization is based on the minimum and maximum with the following shift and scale:

  ```
  shift = 0
  scale = max{abs(max), abs(min)}
  ```

  See "INSERT_NORM_LIN_SCALE Procedure".
- Z-Score Normalization

  Normalization is based on the mean and standard deviation with the following shift and scale:

  ```
  shift = mean
  scale = standard_deviation
  ```

  See "INSERT_NORM_LIN_ZSCORE Procedure".

Outlier Treatment

An outlier is a numerical value that is located far from the rest of the data. Outliers can artificially skew the results of machine learning.

- Winsorizing

  Outliers are replaced with the nearest value that is not an outlier.

  See "INSERT_CLIP_WINSOR_TAIL Procedure"
- Trimming

  Outliers are set to `NULL`.

  See "INSERT_CLIP_TRIM_TAIL Procedure".

Missing Value Treatment

Missing data may indicate sparsity or it may indicate that some values are missing at random. `DBMS_DATA_MINING_TRANSFORM` supports the following transformations for minimizing the effects of missing values:

- Missing numerical values are replaced with the mean.

  See "INSERT_MISS_NUM_MEAN Procedure".
- Missing categorical values are replaced with the mode.

  See "INSERT_MISS_CAT_MODE Procedure".

Note:

Oracle Machine Learning for SQL also has default mechanisms for handling missing data. See *Oracle Machine Learning for SQL User's Guide* for details.

42.2.1.2 DBMS_DATA_MINING_TRANSFORM Security Model

The `DBMS_DATA_MINING_TRANSFORM` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The `DBMS_DATA_MINING_TRANSFORM.INSERT_*` procedures have a *`data_table_name`* parameter that enables the user to provide the input data for transformation purposes. The value of *`data_table_name`* can be the name of a physical table or a view. The *`data_table_name`* parameter can also accept an inline query.

Note:

Because an inline query can be used to specify the data for transformation, Oracle strongly recommends that the calling routine perform any necessary SQL injection checks on the input string.

See Also:

"Operational Notes" for a description of the `DBMS_DATA_MINING_TRANSFORM.INSERT_*` procedures

42.2.1.3 DBMS_DATA_MINING_TRANSFORM Datatypes

`DBMS_DATA_MINING_TRANSFORM` defines the datatypes described in the following table.

Table 42-123 Datatypes in DBMS_DATA_MINING_TRANSFORM

List Type	List Elements	Description
COLUMN_LIST	`VARRAY(1000) OF varchar2(32)`	`COLUMN_LIST` stores quoted and non-quoted identifiers for column names. `COLUMN_LIST` is the datatype of the *`exclude_list`* parameter in the `INSERT` procedures. See "INSERT_AUTOBIN_NUM_EQWIDTH Procedure" for an example. See *Oracle Database PL/SQL Language Reference* for information about populating `VARRAY` structures.

Table 42-123 (Cont.) Datatypes in DBMS_DATA_MINING_TRANSFORM

List Type	List Elements	Description

DESCRIBE_ LIST	`DBMS_SQL.DESC_TAB2`
	
	`TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER`
	
	`TYPE desc_rec2 IS RECORD (`
	`col_type BINARY_INTEGER := 0,`
	`col_max_len BINARY_INTEGER := 0,`
	`col_name VARCHAR2(32767):= '',`
	`col_name_len BINARY_INTEGER := 0,`
	`col_schema_name VARCHAR2(32)    := '',`
	`col_schema_name_len BINARY_INTEGER := 0,`
	`col_precision BINARY_INTEGER := 0,`
	`col_scale BINARY_INTEGER := 0,`
	`col_charsetid BINARY_INTEGER := 0,`
	`col_charsetform BINARY_INTEGER := 0,`
	`col_null_ok         BOOLEAN := TRUE);`
		`DESCRIBE_LIST` describes the columns of the data table after the transformation list has been applied. A `DESCRIBE_LIST` is returned by the DESCRIBE_STACK Procedure.
		The `DESC_TAB2` and `DESC_REC2` types are defined in the `DBMS_SQL` package. See "DESC_REC2 Record Type".
		The `col_type` field of `DESC_REC2` identifies the datatype of the column. The datatype is expressed as a numeric constant that represents a built-in datatype. For example, a 1 indicates a variable length character string. The codes for Oracle built-in datatypes are listed in *Oracle Database SQL Language Reference*. The codes for the Oracle Machine Learning for SQL nested types are described in "Constants".
		The `col_name` field of `DESC_REC2` identifies the column name. It may be populated with a column name, an alias, or an expression. If the column name is a `SELECT` expression, it may be very long. If the expression is longer than 30 bytes, it cannot be used in a view unless it is given an alias.

Table 42-123 (Cont.) Datatypes in DBMS_DATA_MINING_TRANSFORM

List Type	List Elements	Description
TRANSFORM_LIST	**TABLE OF transform_rec** **TYPE transform_rec IS RECORD** (attribute_name VARCHAR2(30), attribute_subname VARCHAR2(4000), expression **EXPRESSION_REC,** reverse_expression **EXPRESSION_REC,** attribute_spec VARCHAR2(4000)); **TYPE expression_rec IS RECORD** (lstmt DBMS_SQL.VARCHAR2A, lb BINARY_INTEGER DEFAULT 1, ub BINARY_INTEGER DEFAULT 0); TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;	TRANSFORM_LIST is a list of transformations that can be embedded in a model. A TRANSFORM_LIST is accepted as an argument by the CREATE_MODEL Procedure. Each element in a TRANSFORM_LIST is a TRANSFORM_REC that specifies how to transform a single attribute. The attribute_name is a column name. The attribute_subname is the nested attribute name if the column is nested, otherwise attribute_subname is null. The expression field holds a SQL expression for transforming the attribute. See "About Transformation Lists" for an explanation of reverse expressions. The attribute_spec field can be used to cause the attribute to be handled in a specific way during the model build. See Table 42-155 for details. The expressions in a TRANSFORM_REC have type EXPRESSION_REC. The lstmt field stores a VARCHAR2A, which is a table of VARCHAR2(32767). The VARCHAR2A datatype allows transformation expressions to be very long, as they can be broken up across multiple rows of VARCHAR2. The VARCHAR2A type is defined in the DBMS_SQL package. See "VARCHAR2A Table Type". The ub (upper bound) and lb (lower bound) fields indicate how many rows there are in the VARCHAR2A table. If ub < lb (default) the EXPRESSION_REC is empty; if lb=ub=1 there is one row; if lb=1 and ub=2 there are 2 rows, and so on.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

42.2.1.4 DBMS_DATA_MINING_TRANSFORM Constants

DBMS_DATA_MINING_TRANSFORM defines the constants described in the following table.

Table 42-124 Constants in DBMS_DATA_MINING_TRANSFORM

Constant	Value	Description

`NEST_NUM_COL_TYPE`	100001	Indicates that an attribute in the transformation list comes from a row in a column of `DM_NESTED_NUMERICALS`.
		Nested numerical attributes are defined as follows:
		`attribute_name VARCHAR2(4000)`
		`value NUMBER`

`NEST_CAT_COL_TYPE`	100002	Indicates that an attribute in the transformation list comes from a row in a column of `DM_NESTED_CATEGORICALS`.
		Nested categorical attributes are defined as follows:
		`attribute_name VARCHAR2(4000)`
		`value VARCHAR2(4000)`

`NEST_BD_COL_TYPE`	100003	Indicates that an attribute in the transformation list comes from a row in a column of `DM_NESTED_BINARY_DOUBLES`.
		Nested binary double attributes are defined as follows:
		`attribute_name VARCHAR2(4000)`
		`value BINARY_DOUBLE`

`NEST_BF_COL_TYPE`	100004	Indicates that an attribute in the transformation list comes from a row in a column of `DM_NESTED_BINARY_FLOATS`.
		`attribute_name VARCHAR2(4000)`
		`value BINARY_FLOAT`

See Also:

Oracle Machine Learning for SQL User’s Guide for information about nested data in Oracle Machine Learning for SQL

42.2.2 DBMS_DATA_MINING_TRANSFORM Operational Notes

The `DBMS_DATA_MINING_TRANSFORM` package offers a flexible framework for specifying data transformations. If you choose to embed transformations in the model (the preferred method), you create a **transformation list** object and pass it to the CREATE_MODEL Procedure. If you choose to transform the data without embedding, you create a view.

When specified in a transformation list, the transformation expressions are run by the model. When specified in a view, the transformation expressions are run by the view.

Transformation Definitions

Transformation definitions are used to generate the SQL expressions that transform the data. For example, the transformation definitions for normalizing a numeric column are the shift and scale values for that data.

With the `DBMS_DATA_MINING_TRANSFORM` package, you can call procedures to compute the transformation definitions, or you can compute them yourself, or you can do both.

Transformation Definition Tables

`DBMS_DATA_MINING_TRANSFORM` provides **`INSERT`** procedures that compute transformation definitions and insert them in transformation definition tables. You can modify the values in the transformation definition tables or populate them yourself.

`XFORM` routines use populated definition tables to transform data in external views. **`STACK`** routines use populated definition tables to build transformation lists.

To specify transformations based on definition tables, follow these steps:

1. Use **`CREATE`** routines to create transformation definition tables.

   The tables have columns to hold the transformation definitions for a given type of transformation. For example, the CREATE_BIN_NUM Procedure creates a definition table that has a column for storing data values and another column for storing the associated bin identifiers.

2. Use **`INSERT`** routines to compute and insert transformation definitions in the tables.

   Each `INSERT` routine uses a specific technique for computing the transformation definitions. For example, the INSERT_BIN_NUM_EQWIDTH Procedure computes bin boundaries by identifying the minimum and maximum values then setting the bin boundaries at equal intervals.

3. Use **`STACK`** or **`XFORM`** routines to generate transformation expressions based on the information in the definition tables:

   - Use **`STACK`** routines to add the transformation expressions to a transformation list. Pass the transformation list to the CREATE_MODEL Procedure. The transformation expressions will be assembled into one long SQL query and embedded in the model.
   - Use **`XFORM`** routines to execute the transformation expressions within a view. The transformations will be external to the model and will need to be re-created whenever the model is applied to new data.

Transformations Without Definition Tables

`STACK` routines are not the only method for adding transformation expressions to a transformation list. You can also build a transformation list without using definition tables.

To specify transformations without using definition tables, follow these steps:

1. Write a SQL expression for transforming an attribute.
2. Write a SQL expression for reversing the transformation. (See "Reverse Transformations and Model Transparency" in "DBMS_DATA_MINING_TRANSFORM-About Transformation Lists".)
3. Determine whether or not to disable ADP for the attribute. By default ADP is enabled for the attribute if it is specified for the model. (See "Disabling Automatic Data Preparation" in "DBMS_DATA_MINING_TRANSFORM - About Transformation Lists".)
4. Specify the SQL expressions and ADP instructions in a call to the SET_TRANSFORM Procedure, which adds the information to a transformation list.
5. Repeat steps 1 through 4 for each attribute that you wish to transform.

6. Pass the transformation list to the CREATE_MODEL Procedure. The transformation expressions will be assembled into one long SQL query and embedded in the model.

> **Note:**
>
> SQL expressions that you specify with `SET_TRANSFORM` must fit within a `VARCHAR2`. To specify a longer expression, you can use the SET_EXPRESSION Procedure. With `SET_EXPRESSION`, you can build an expression by appending rows to a `VARCHAR2` array.

About Stacking

Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.

Related Topics

- CREATE_MODEL Procedure
  This procedure creates an Oracle Machine Learning for SQL model with a given machine learning function.
- DBMS_DATA_MINING_TRANSFORM — About Transformation Lists
  The elements of a transformation list are **transformation records**. Each transformation record provides all the information needed by the model for managing the transformation of a single attribute.
- DBMS_DATA_MINING_TRANSFORM — About Stacking and Stack Procedures
  Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.
- DBMS_DATA_MINING_TRANSFORM — Nested Data Transformations
  The `CREATE` routines create transformation definition tables that include two columns, `col` and `att`, for identifying attributes.

42.2.2.1 DBMS_DATA_MINING_TRANSFORM — About Transformation Lists

The elements of a transformation list are **transformation records**. Each transformation record provides all the information needed by the model for managing the transformation of a single attribute.

Each transformation record includes the following fields:

- *`attribute_name`* — Name of the column of data to be transformed
- *`attribute_subname`* — Name of the nested attribute if *`attribute_name`* is a nested column, otherwise `NULL`
- *`expression`* — SQL expression for transforming the attribute
- *`reverse_expression`* — SQL expression for reversing the transformation
- *`attribute_spec`* — Identifies special treatment for the attribute during the model build. See Table 42-155 for details.

See Also:

- Table 42-123 for details about the TRANSFORM_LIST and TRANSFORM_REC object types
- SET_TRANSFORM Procedure
- CREATE_MODEL Procedure

Reverse Transformations and Model Transparency

An algorithm manipulates transformed attributes to train and score a model. The transformed attributes, however, may not be meaningful to an end user. For example, if attribute *x* has been transformed into bins 1 — 4, the bin names 1, 2 , 3, and 4 are manipulated by the algorithm, but a user is probably not interested in the model details about bins 1 — 4 or in predicting the numbers 1 — 4.

To return original attribute values in model details and predictions, you can provide a reverse expression in the transformation record for the attribute. For example, if you specify the transformation expression **'log(10, y)'** for attribute *y*, you could specify the reverse transformation expression **'power(10, y)'**.

Reverse transformations enable **model transparency**. They make internal processing transparent to the user.

Note:

STACK procedures automatically reverse normalization transformations, but they do not provide a mechanism for reversing binning, clipping, or missing value transformations.

You can use the DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION procedure to specify or update reverse transformations expressions for an existing model.

See Also:

Table 42-123

"ALTER_REVERSE_EXPRESSION Procedure"

"Summary of DBMS_DATA_MINING Subprograms" for links to the model details functions

Disabling Automatic Data Preparation

ADP is controlled by a model-specific setting (PREP_AUTO). The PREP_AUTO setting affects all model attributes unless you disable it for individual attributes.

If ADP is enabled and you set `attribute_spec` to `NOPREP`, only the transformations that you specify for that attribute will be evaluated. If ADP is enabled and you do *not* set `attribute_spec` to `NOPREP`, the automatic transformations will be evaluated *after* the transformations that you specify for the attribute.

If ADP is not enabled for the model, the `attribute_spec` field of the transformation record is ignored.

> **See Also:**
>
> "Automatic Data Preparation" for information about the `PREP_AUTO` setting

Adding Transformation Records to a Transformation List

A transformation list is a stack of transformation records. When a new transformation record is added, it is appended to the top of the stack. (See "About Stacking" for details.)

When you use `SET_TRANSFORM` to add a transformation record to a transformation list, you can specify values for all the fields in the transformation record.

When you use `STACK` procedures to add transformation records to a transformation list, only the transformation expression field is populated. For normalization transformations, the reverse transformation expression field is also populated.

You can use both `STACK` procedures and `SET_TRANSFORM` to build one transformation list. Each `STACK` procedure call adds transformation records for all the attributes in a specified transformation definition table. Each `SET_TRANSFORM` call adds a transformation record for a single attribute.

42.2.2.2 DBMS_DATA_MINING_TRANSFORM — About Stacking and Stack Procedures

Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.

Stack Procedures

`STACK` procedures create transformation records from the information in transformation definition tables. For example `STACK_BIN_NUM` builds a transformation record for each attribute specified in a definition table for numeric binning. `STACK` procedures stack the transformation records as follows:

- If an attribute is specified in the definition table but not in the transformation list, the `STACK` procedure creates a transformation record, computes the reverse transformation (if possible), inserts the transformation and reverse transformation in the transformation record, and appends the transformation record to the top of the transformation list.
- If an attribute is specified in the transformation list but not in the definition table, the `STACK` procedure takes no action.

- If an attribute is specified in the definition table *and* in the transformation list, the STACK procedure stacks the transformation expression from the definition table on top of the transformation expression in the transformation record and updates the reverse transformation. See Table 42-123and Example 42-6.

Example 42-3   Stacking a Clipping Transformation

This example shows how STACK_CLIP Procedure would add transformation records to a transformation list. Note that the clipping transformations are not reversed in COL1 and COL2 after stacking (as described in "Reverse Transformations and Model Transparency" in "DBMS_DATA_MINING_TRANSFORM-About Transformation Lists").

Refer to:

- CREATE_CLIP Procedure — Creates the definition table
- INSERT_CLIP_TRIM_TAIL Procedure — Inserts definitions in the table
- INSERT_CLIP_WINSOR_TAIL Procedure — Inserts definitions in the table
- Table 42-123 — Describes the structure of the transformation list (TRANSFORM_LIST object)

Assume a clipping definition table populated as follows.

col	att	lcut	lval	rcut	rval
COL1	null	-1.5	-1.5	4.5	4.5
COL2	null	0	0	1	1

Assume the following transformation list before stacking.

```
-------------------------
transformation record #1:
-------------------------
     attribute_name       = COL1
     attribute_subname    = null
     expression           = log(10, COL1)
     reverse_expression   = power(10, COL1)
-------------------------
transformation record #2:
-------------------------
     attribute_name       = COL3
     attribute_subname    = null
     expression           = ln(COL3)
     reverse_expression   = exp(COL3)
```

After stacking, the transformation list is as follows.

```
-------------------------
transformation record #1:
-------------------------
     attribute_name       = COL1
     attribute_subname    = null
     expression           = CASE WHEN log(10, COL1) < -1.5 THEN -1.5
                                 WHEN log(10, COL1) > 4.5  THEN 4.5
                                 ELSE log(10, COL1)
                            END;
     reverse_expression   = power(10, COL1)
-------------------------
transformation record #2:
```

```
-------------------------
     attribute_name       = COL3
     attribute_subname    = null
     expression           = ln(COL3)
     reverse_expression   = exp(COL3)
-------------------------
transformation record #3:
-------------------------
     attribute_name       = COL2
     attribute_subname    = null
     expression           = CASE WHEN COL2 < 0 THEN 0
                                 WHEN COL2 > 1 THEN 1
                                 ELSE COL2
                            END;
     reverse_expression   = null
```

42.2.2.3 DBMS_DATA_MINING_TRANSFORM — Nested Data Transformations

The `CREATE` routines create transformation definition tables that include two columns, `col` and `att`, for identifying attributes.

The column `col` holds the name of a column in the data table. If the data column is not nested, then `att` is null, and the name of the attribute is *`col`*. If the data column is nested, then `att` holds the name of the nested attribute, and the name of the attribute is *`col.att`*. The `INSERT` and `XFORM` routines ignore the `att` column in the definition tables. Neither the `INSERT` nor the `XFORM` routines support nested data.

Only the `STACK` procedures and `SET_TRANSFORM` support nested data. Nested data transformations are always embedded in the model.

Nested columns in Oracle Machine Learning for SQL can have the following types:

```
DM_NESTED_NUMERICALS
DM_NESTED_CATEGORICALS
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
```

> **See Also:**
>
> "Constants"
>
> *Oracle Machine Learning for SQL User's Guide* for details about nested attributes in Oracle Machine Learning for SQL

Specifying Nested Attributes in a Transformation Record

A transformation record (`TRANSFORM_REC`) includes two fields, `attribute_name` and `attribute_subname`, for identifying the attribute. The field `attribute_name` holds the name of a column in the data table. If the data column is not nested, then `attribute_subname` is null, and the name of the attribute is *`attribute_name`*. If the data column is nested, then `attribute_subname` holds the name of the nested attribute, and the name of the attribute is *`attribute_name.attribute_subname`*.

Transforming Individual Nested Attributes

You can specify different transformations for different attributes in a nested column, and you can specify a default transformation for all the remaining attributes in the column. To specify a default nested transformation, specify null in the `attribute_name` field and the name of the nested column in the `attribute_subname` field as shown in Example 42-4. Note that the keyword `VALUE` is used to represent the value of a nested attribute in a transformation expression.

Example 42-4 Transforming a Nested Column

The following statement transforms two of the nested attributes in `COL_N1`. Attribute `ATTR1` is transformed with normalization; Attribute `ATTR2` is set to null, which causes attribute removal transformation (`ATTR2` is not used in training the model). All the remaining attributes in `COL_N1` are divided by 10.

```
DECLARE
  stk dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(
      stk,'COL_N1', 'ATTR1', '(VALUE - (-1.5))/20', 'VALUE *20 + (-1.5)');
  dbms_data_mining_transform.SET_TRANSFORM(
      stk,'COL_N1', 'ATTR2', NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM(
      stk, NULL, 'COL_N1', 'VALUE/10', 'VALUE*10');
END;
/
```

The following SQL is generated from this statement.

```
CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
                               "ATTRIBUTE_NAME",
                               DECODE("ATTRIBUTE_NAME",
                                 'ATTR1', ("VALUE" - (-1.5))/20,
                                 "VALUE"/10))
                        FROM TABLE("COL_N1")
                       WHERE "ATTRIBUTE_NAME" IS NOT IN ('ATTR2'))
             AS DM_NESTED_NUMERICALS)
```

If transformations are not specified for `COL_N1.ATTR1` and `COL_N1.ATTR2`, then the default transformation is used for all the attributes in `COL_N1`, and the resulting SQL does not include a `DECODE`.

```
    CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
                               "ATTRIBUTE_NAME",
                               "VALUE"/10)
                        FROM TABLE("COL_N1"))
             AS DM_NESTED_NUMERICALS)
```

Since `DECODE` is limited to 256 arguments, multiple `DECODE` functions are nested to support an arbitrary number of individual nested attribute specifications.

Adding a Nested Column

You can specify a transformation that adds a nested column to the data, as shown in Example 42-5.

Example 42-5 Adding a Nested Column to a Transformation List

```
DECLARE
    v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
      'YOB_CREDLIM', NULL,
      'dm_nested_numericals(
           dm_nested_numerical(
                 ''CUST_YEAR_OF_BIRTH'', cust_year_of_birth),
           dm_nested_numerical(
                 ''CUST_CREDIT_LIMIT'', cust_credit_limit))',
        NULL);
    dbms_data_mining_transform.SET_TRANSFORM(
              v_xlst, 'CUST_YEAR_OF_BIRTH', NULL, NULL, NULL);
    dbms_data_mining_transform.SET_TRANSFORM(
              v_xlst, 'CUST_CREDIT_LIMIT', NULL, NULL, NULL);
    dbms_data_mining_transform.XFORM_STACK(
              v_xlst, 'mining_data', 'mining_data_v');
END;
/

set long 2000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_V';

TEXT
---------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_POSTAL_CODE",dm_nested_numericals(
        dm_nested_numerical(
           'CUST_YEAR_OF_BIRTH', cust_year_of_birth),
        dm_nested_numerical(
           'CUST_CREDIT_LIMIT', cust_credit_limit)) "YOB_CREDLIM" FROM
mining_data

SELECT * FROM mining_data_v WHERE cust_id = 104500;

CUST_ID CUST_POSTAL_CODE YOB_CREDLIM(ATTRIBUTE_NAME, VALUE)
------- ----------------
-----------------------------------------------------
 104500 68524            DM_NESTED_NUMERICALS(DM_NESTED_NUMERICAL(
                        'CUST_YEAR_OF_BIRTH', 1962),
                         DM_NESTED_NUMERICAL('CUST_CREDIT_LIMIT', 15000))
```

Stacking Nested Transformations

Example 42-6 shows how the STACK_NORM_LIN Procedure would add transformation records for nested column `COL_N` to a transformation list.

Refer to:

- CREATE_NORM_LIN Procedure — Creates the definition table
- INSERT_NORM_LIN_MINMAX Procedure — Inserts definitions in the table
- INSERT_NORM_LIN_SCALE Procedure — Inserts definitions in the table
- INSERT_NORM_LIN_ZSCORE Procedure — Inserts definitions in the table
- Table 42-123 — Describes the structure of the transformation list

Example 42-6 Stacking a Nested Normalization Transformation

Assume a linear normalization definition table populated as follows.

col	att	shift	scale
COL_N	ATT2	0	20
null	COL_N	0	10

Assume the following transformation list before stacking.

```
-------------------------
transformation record #1:
-------------------------
     attribute_name       = COL_N
     attribute_subname    = ATT1
     expression           = log(10, VALUE)
     reverse_expression   = power(10, VALUE)
-------------------------
transformation record #2:
-------------------------
     attribute_name       = null
     attribute_subname    = COL_N
     expression           = ln(VALUE)
     reverse_expression   = exp(VALUE)
```

After stacking, the transformation list is as follows.

```
-------------------------
transformation record #1:
-------------------------
     attribute_name       = COL_N
     attribute_subname    = ATT1
     expression           = (log(10, VALUE) - 0)/10
     reverse_expression   = power(10, VALUE*10 + 0)
-------------------------
transformation record #2:
-------------------------
     attribute_name       = NULL
     attribute_subname    = COL_N
     expression           = (ln(VALUE)- 0)/10
     reverse_expression   = exp(VALUE *10 + 0)
-------------------------
transformation record #3:
-------------------------
     attribute_name       = COL_N
     attribute_subname    = ATT2
     expression           = (ln(VALUE) - 0)/20
     reverse_expression   = exp(VALUE * 20 + 0)
```

42.2.3 Summary of DBMS_DATA_MINING_TRANSFORM Subprograms

This table lists the `DBMS_DATA_MINING_TRANSFORM` subprograms in alphabetical order and briefly describes them.

Table 42-125 DBMS_DATA_MINING_TRANSFORM Package Subprograms

Subprogram	Purpose
CREATE_BIN_CAT Procedure	Creates a transformation definition table for categorical binning

Table 42-125 (Cont.) DBMS_DATA_MINING_TRANSFORM Package Subprograms

Subprogram	Purpose
CREATE_BIN_NUM Procedure	Creates a transformation definition table for numerical binning
CREATE_CLIP Procedure	Creates a transformation definition table for clipping
CREATE_COL_REM Procedure	Creates a transformation definition table for column removal
CREATE_MISS_CAT Procedure	Creates a transformation definition table for categorical missing value treatment
CREATE_MISS_NUM Procedure	Creates a transformation definition table for numerical missing values treatment
CREATE_NORM_LIN Procedure	Creates a transformation definition table for linear normalization
DESCRIBE_STACK Procedure	Describes the transformation list
GET_EXPRESSION Function	Returns a `VARCHAR2` chunk from a transformation expression
INSERT_AUTOBIN_NUM_EQWIDTH Procedure	Inserts numeric automatic equi-width binning definitions in a transformation definition table
INSERT_BIN_CAT_FREQ Procedure	Inserts categorical frequency-based binning definitions in a transformation definition table
INSERT_BIN_NUM_EQWIDTH Procedure	Inserts numeric equi-width binning definitions in a transformation definition table
INSERT_BIN_NUM_QTILE Procedure	Inserts numeric quantile binning expressions in a transformation definition table
INSERT_BIN_SUPER Procedure	Inserts supervised binning definitions in numerical and categorical transformation definition tables
INSERT_CLIP_TRIM_TAIL Procedure	Inserts numerical trimming definitions in a transformation definition table
INSERT_CLIP_WINSOR_TAIL Procedure	Inserts numerical winsorizing definitions in a transformation definition table
INSERT_MISS_CAT_MODE Procedure	Inserts categorical missing value treatment definitions in a transformation definition table
INSERT_MISS_NUM_MEAN Procedure	Inserts numerical missing value treatment definitions in a transformation definition table
INSERT_NORM_LIN_MINMAX Procedure	Inserts linear min-max normalization definitions in a transformation definition table
INSERT_NORM_LIN_SCALE Procedure	Inserts linear scale normalization definitions in a transformation definition table
INSERT_NORM_LIN_ZSCORE Procedure	Inserts linear zscore normalization definitions in a transformation definition table
SET_EXPRESSION Procedure	Adds a `VARCHAR2` chunk to an expression
SET_TRANSFORM Procedure	Adds a transformation record to a transformation list
STACK_BIN_CAT Procedure	Adds a categorical binning expression to a transformation list
STACK_BIN_NUM Procedure	Adds a numerical binning expression to a transformation list
STACK_CLIP Procedure	Adds a clipping expression to a transformation list
STACK_COL_REM Procedure	Adds a column removal expression to a transformation list
STACK_MISS_CAT Procedure	Adds a categorical missing value treatment expression to a transformation list

Table 42-125 (Cont.) DBMS_DATA_MINING_TRANSFORM Package Subprograms

Subprogram	Purpose
STACK_MISS_NUM Procedure	Adds a numerical missing value treatment expression to a transformation list
STACK_NORM_LIN Procedure	Adds a linear normalization expression to a transformation list
XFORM_BIN_CAT Procedure	Creates a view of the data table with categorical binning transformations
XFORM_BIN_NUM Procedure	Creates a view of the data table with numerical binning transformations
XFORM_CLIP Procedure	Creates a view of the data table with clipping transformations
XFORM_COL_REM Procedure	Creates a view of the data table with column removal transformations
XFORM_EXPR_NUM Procedure	Creates a view of the data table with the specified numeric transformations
XFORM_EXPR_STR Procedure	Creates a view of the data table with the specified categorical transformations
XFORM_MISS_CAT Procedure	Creates a view of the data table with categorical missing value treatment
XFORM_MISS_NUM Procedure	Creates a view of the data table with numerical missing value treatment
XFORM_NORM_LIN Procedure	Creates a view of the data table with linear normalization transformations
XFORM_STACK Procedure	Creates a view of the transformation list

42.2.3.1 CREATE_BIN_CAT Procedure

This procedure creates a transformation definition table for categorical binning.

The columns are described in the following table.

Table 42-126 Columns in a Transformation Definition Table for Categorical Binning

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of categorical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is a nested column. If *`col`* is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.
`val`	`VARCHAR2(4000)`	Values of the attribute
`bin`	`VARCHAR2(4000)`	Bin assignments for the values

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT (
     bin_table_name      IN VARCHAR2,
     bin_schema_name     IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-127 CREATE_BIN_CAT Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table to be created
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User’s Guide* for details about categorical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:
   - INSERT_BIN_CAT_FREQ Procedure — frequency-based binning
   - INSERT_BIN_SUPER Procedure — supervised binning

> **See Also:**
>
> "Binning" in DBMS_DATA_MINING_TRANSFORM Overview
>
> "Operational Notes"

Examples

The following statement creates a table called `bin_cat_xtbl` in the current schema. The table has columns that can be populated with bin assignments for categorical attributes.

```
BEGIN
   DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT('bin_cat_xtbl');
END;
/
DESCRIBE bin_cat_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 VAL                                                VARCHAR2(4000)
 BIN                                                VARCHAR2(4000)
```

42.2.3.2 CREATE_BIN_NUM Procedure

This procedure creates a transformation definition table for numerical binning.

The columns are described in the following table.

Table 42-128 Columns in a Transformation Definition Table for Numerical Binning

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is a nested column. If *`col`* is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.
`val`	`NUMBER`	Values of the attribute
`bin`	`VARCHAR2(4000)`	Bin assignments for the values

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM (
     bin_table_name    IN VARCHAR2,
     bin_schema_name   IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-129 CREATE_BIN_NUM Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table to be created
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:
   - INSERT_AUTOBIN_NUM_EQWIDTH Procedure — automatic equi-width binning
   - INSERT_BIN_NUM_EQWIDTH Procedure — user-specified equi-width binning
   - INSERT_BIN_NUM_QTILE Procedure — quantile binning
   - INSERT_BIN_SUPER Procedure — supervised binning

See Also:

"Binning" in DBMS_DATA_MINING_TRANSFORM Overview

"Operational Notes"

Examples

The following statement creates a table called `bin_num_xtbl` in the current schema. The table has columns that can be populated with bin assignments for numerical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM('bin_num_xtbl');
END;
/

DESCRIBE bin_num_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 VAL                                                NUMBER
 BIN                                                VARCHAR2(4000)
```

42.2.3.3 CREATE_CLIP Procedure

This procedure creates a transformation definition table for clipping or winsorizing to minimize the effect of outliers.

The columns are described in the following table.

Table 42-130 Columns in a Transformation Definition Table for Clipping or Winsorizing

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is a nested column of `DM_NESTED_NUMERICALS`. If *`col`* is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.
`lcut`	`NUMBER`	The lowest typical value for the attribute. If the attribute values were plotted on an *xy* axis, *`lcut`* would be the left-most boundary of the range of values considered typical for this attribute. Any values to the left of *`lcut`* are outliers.
`lval`	`NUMBER`	Value assigned to an outlier to the left of *`lcut`*

Table 42-130 (Cont.) Columns in a Transformation Definition Table for Clipping or Winsorizing

Name	Datatype	Description
`rcut`	`NUMBER`	The highest typical value for the attribute If the attribute values were plotted on an *xy* axis, *`rcut`* would be the right-most boundary of the range of values considered typical for this attribute. Any values to the right of *`rcut`* are outliers.
`rval`	`NUMBER`	Value assigned to an outlier to the right of *`rcut`*

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP (
     clip_table_name     IN VARCHAR2,
     clip_schema_name    IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-131 CREATE_CLIP Procedure Parameters

Parameter	Description
`clip_table_name`	Name of the transformation definition table to be created
`clip_schema_name`	Schema of *`clip_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:
   - INSERT_CLIP_TRIM_TAIL Procedure — replaces outliers with nulls
   - INSERT_CLIP_WINSOR_TAIL Procedure — replaces outliers with an average value

> **See Also:**
>
> "Outlier Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
>
> "Operational Notes"

Examples

The following statement creates a table called `clip_xtbl` in the current schema. The table has columns that can be populated with clipping instructions for numerical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP('clip_xtbl');
```

```
END;
/

DESCRIBE clip_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 LCUT                                               NUMBER
 LVAL                                               NUMBER
 RCUT                                               NUMBER
 RVAL                                               NUMBER
```

42.2.3.4 CREATE_COL_REM Procedure

This procedure creates a transformation definition table for removing columns from the data table.

The columns are described in the following table.

Table 42-132 Columns in a Transformation Definition Table for Column Removal

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is nested (`DM_NESTED_NUMERICALS` or `DM_NESTED_CATEGORICALS`). If col is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM (
      rem_table_name           VARCHAR2,
      rem_schema_name          VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-133 CREATE_COL_REM Procedure Parameters

Parameter	Description
`rem_table_name`	Name of the transformation definition table to be created
`rem_schema_name`	Schema of *`rem_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See "Nested Data Transformations" for information about transformation definition tables and nested data.
2. See "Operational Notes".

Examples

The following statement creates a table called `rem_att_xtbl` in the current schema. The table has columns that can be populated with the names of attributes to exclude from the data to be mined.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('rem_att_xtbl');
END;
 /
DESCRIBE rem_att_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
```

42.2.3.5 CREATE_MISS_CAT Procedure

This procedure creates a transformation definition table for replacing categorical missing values.

The columns are described in the following table.

Table 42-134 Columns in a Transformation Definition Table for Categorical Missing Value Treatment

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of categorical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is a nested column of `DM_NESTED_CATEGORICALS`. If *`col`* is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.
`val`	`VARCHAR2(4000)`	Replacement for missing values in the attribute

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT (
     miss_table_name        IN VARCHAR2,
     miss_schema_name       IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-135 CREATE_MISS_CAT Procedure Parameters

Parameter	Description
`miss_table_name`	Name of the transformation definition table to be created
`miss_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about categorical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the INSERT_MISS_CAT_MODE Procedure to populate the transformation definition table.

> **See Also:**
>
> "Missing Value Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
>
> "Operational Notes"

Examples

The following statement creates a table called `miss_cat_xtbl` in the current schema. The table has columns that can be populated with values for missing data in categorical attributes.

```
BEGIN

  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('miss_cat_xtbl');
END;
/

DESCRIBE miss_cat_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 VAL                                                VARCHAR2(4000)
```

42.2.3.6 CREATE_MISS_NUM Procedure

This procedure creates a transformation definition table for replacing numerical missing values.

The columns are described in Table 42-136.

Table 42-136 Columns in a Transformation Definition Table for Numerical Missing Value Treatment

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.

Table 42-136 (Cont.) Columns in a Transformation Definition Table for Numerical Missing Value Treatment

Name	Datatype	Description
att	VARCHAR2(4000)	The attribute subname if *col* is a nested column of DM_NESTED_NUMERICALS. If *col* is nested, the attribute name is *col.att*. If *col* is not nested, *att* is null.
val	NUMBER	Replacement for missing values in the attribute

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM (
     miss_table_name       IN VARCHAR2,
     miss_schema_name      IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-137 CREATE_MISS_NUM Procedure Parameters

Parameter	Description
miss_table_name	Name of the transformation definition table to be created
miss_schema_name	Schema of *miss_table_name*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the INSERT_MISS_NUM_MEAN Procedure to populate the transformation definition table.

> **See Also:**
>
> "Missing Value Treatment" in DBMS_DATA_MINING_TRANSFORM Overview
>
> "Operational Notes"

Example

The following statement creates a table called miss_num_xtbl in the current schema. The table has columns that can be populated with values for missing data in numerical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('miss_num_xtbl');
END;
/
```

```
DESCRIBE miss_num_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 VAL                                                NUMBER
```

42.2.3.7 CREATE_NORM_LIN Procedure

This procedure creates a transformation definition table for linear normalization.

The columns are described in Table 42-138.

Table 42-138 Columns in a Transformation Definition Table for Linear Normalization

Name	Datatype	Description
`col`	`VARCHAR2(30)`	Name of a column of numerical data. If the column is not nested, the column name is also the attribute name. For information about attribute names, see *Oracle Machine Learning for SQL User's Guide*.
`att`	`VARCHAR2(4000)`	The attribute subname if *`col`* is a nested column of `DM_NESTED_NUMERICALS`. If *`col`* is nested, the attribute name is *`col.att`*. If *`col`* is not nested, *`att`* is null.
`shift`	`NUMBER`	A constant to subtract from the attribute values
`scale`	`NUMBER`	A constant by which to divide the shifted values

Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN (
     norm_table_name       IN VARCHAR2,
     norm_schema_name      IN VARCHAR2 DEFAULT NULL );
```

Parameters

Table 42-139 CREATE_NORM_LIN Procedure Parameters

Parameter	Description
`norm_table_name`	Name of the transformation definition table to be created
`norm_schema_name`	Schema of *`norm_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. See "Nested Data Transformations" for information about transformation definition tables and nested data.
3. You can use the following procedures to populate the transformation definition table:

- INSERT_NORM_LIN_MINMAX Procedure — Uses linear min-max normalization
- INSERT_NORM_LIN_SCALE Procedure — Uses linear scale normalization
- INSERT_NORM_LIN_ZSCORE Procedure — Uses linear zscore normalization

> **See Also:**
>
> "Linear Normalization" in DBMS_DATA_MINING_TRANSFORM Overview
>
> "Operational Notes"

Examples

The following statement creates a table called `norm_xtbl` in the current schema. The table has columns that can be populated with shift and scale values for normalizing numerical attributes.

```
BEGIN
     DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN('norm_xtbl');
END;
/

DESCRIBE norm_xtbl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL                                                VARCHAR2(30)
 ATT                                                VARCHAR2(4000)
 SHIFT                                              NUMBER
 SCALE                                              NUMBER
```

42.2.3.8 DESCRIBE_STACK Procedure

This procedure describes the columns of the data table after a list of transformations has been applied.

Only the columns that are specified in the transformation list are transformed. The remaining columns in the data table are included in the output without changes.

To create a view of the data table after the transformations have been applied, use the XFORM_STACK Procedure.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.DESCRIBE_STACK (
     xform_list           IN   TRANSFORM_LIST,
     data_table_name      IN   VARCHAR2,
     describe_list        OUT  DESCRIBE_LIST,
     data_schema_name     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-140 DESCRIBE_STACK Procedure Parameters

Parameter	Description
xform_list	A list of transformations. See Table 42-123 for a description of the TRANSFORM_LIST object type.
data_table_name	Name of the table containing the data to be transformed
describe_list	Descriptions of the columns in the data table after the transformations specified in *xform_list* have been applied. See Table 42-123 for a description of the DESCRIBE_LIST object type.
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes" for information about transformation lists and embedded transformations.

Examples

This example shows the column name and datatype, the column name length, and the column maximum length for the view oml_user.cust_info after the transformation list has been applied. All the transformations are user-specified. The results of DESCRIBE_STACK do not include one of the columns in the original table, because the SET_TRANSFORM procedure sets that column to NULL.

```
CREATE OR REPLACE VIEW cust_info AS
         SELECT a.cust_id, c.country_id, c.cust_year_of_birth,
         CAST(COLLECT(DM_Nested_Numerical(
                  b.prod_name, 1))
                AS DM_Nested_Numericals) custprods
                 FROM sh.sales a, sh.products b, sh.customers c
                  WHERE a.prod_id = b.prod_id AND
                        a.cust_id=c.cust_id and
                        a.cust_id between 100001 AND 105000
         GROUP BY a.cust_id, country_id, cust_year_of_birth;

describe cust_info
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 COUNTRY_ID                                NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUSTPRODS                                          SYS.DM_NESTED_NUMERICALS

DECLARE
  cust_stack   dbms_data_mining_transform.TRANSFORM_LIST;
  cust_cols    dbms_data_mining_transform.DESCRIBE_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
     'country_id', NULL, 'country_id/10', 'country_id*10');
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
      'cust_year_of_birth', NULL, NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
      'custprods', 'Mouse Pad', 'value*100', 'value/100');
```

```
  dbms_data_mining_transform.DESCRIBE_STACK(
       xform_list => cust_stack,
       data_table_name => 'cust_info',
       describe_list => cust_cols);
  dbms_output.put_line('====');
  for i in 1..cust_cols.COUNT loop
    dbms_output.put_line('COLUMN_NAME:     '||cust_cols(i).col_name);
    dbms_output.put_line('COLUMN_TYPE:     '||cust_cols(i).col_type);
    dbms_output.put_line('COLUMN_NAME_LEN: '||cust_cols(i).col_name_len);
    dbms_output.put_line('COLUMN_MAX_LEN:  '||cust_cols(i).col_max_len);
    dbms_output.put_line('====');
  END loop;
END;
/
====
COLUMN_NAME:     CUST_ID
COLUMN_TYPE:     2
COLUMN_NAME_LEN: 7
COLUMN_MAX_LEN:  22
====
COLUMN_NAME:     COUNTRY_ID
COLUMN_TYPE:     2
COLUMN_NAME_LEN: 10
COLUMN_MAX_LEN:  22
====
COLUMN_NAME:     CUSTPRODS
COLUMN_TYPE:     100001
COLUMN_NAME_LEN: 9
COLUMN_MAX_LEN:  40
====
```

42.2.3.9 GET_EXPRESSION Function

This function returns a row from a VARCHAR2 array that stores a transformation expression. The array is built by calls to the SET_EXPRESSION Procedure.

The array can be used for specifying SQL expressions that are too long to be used with the SET_TRANSFORM Procedure.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.GET_EXPRESSION (
     expression           IN EXPRESSION_REC,
     chunk_num            IN PLS_INTEGER DEFAULT NULL);
 RETURN VARCHAR2;
```

Parameters

Table 42-141 GET_EXPRESSION Function Parameters

Parameter	Description
`expression`	An expression record (`EXPRESSION_REC`) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a `VARCHAR2` array and index fields for specifying upper and lower boundaries within the array. There are two `EXPRESSION_REC` fields within a transformation record (`TRANSFORM_REC`): one for the transformation expression; the other for the reverse transformation expression. See Table 42-123 for a description of the `EXPRESSION_REC` type.
`chunk`	A `VARCHAR2` chunk (row) to be appended to *`expression`*.

Usage Notes

1. Chunk numbering starts with one. For chunks outside of the range, the return value is null. When a chunk number is null the whole expression is returned as a string. If the expression is too big, a `VALUE_ERROR` is raised.
2. See "About Transformation Lists".
3. See "Operational Notes".

Examples

See the example for the SET_EXPRESSION Procedure.

Related Topics

- SET_EXPRESSION Procedure
  This procedure appends a row to a `VARCHAR2` array that stores a SQL expression.
- SET_TRANSFORM Procedure
  This procedure appends the transformation instructions for an attribute to a transformation list.

42.2.3.10 INSERT_AUTOBIN_NUM_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

`INSERT_AUTOBIN_NUM_EQWIDTH` computes the number of bins separately for each column. If you want to use equi-width binning with the same number of bins for each column, use the INSERT_BIN_NUM_EQWIDTH Procedure.

`INSERT_AUTOBIN_NUM_EQWIDTH` bins all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_AUTOBIN_NUM_EQWIDTH (
     bin_table_name        IN VARCHAR2,
     data_table_name       IN VARCHAR2,
```

```
        bin_num              IN PLS_INTEGER DEFAULT 3,
        max_bin_num          IN PLS_INTEGER DEFAULT 100,
        exclude_list         IN COLUMN_LIST DEFAULT NULL,
        round_num            IN PLS_INTEGER DEFAULT 6,
        sample_size          IN PLS_INTEGER DEFAULT 50000,
        bin_schema_name      IN VARCHAR2 DEFAULT NULL,
        data_schema_name     IN VARCHAR2 DEFAULT NULL,
        rem_table_name       IN VARCHAR2 DEFAULT NULL,
        rem_schema_name      IN VARCHAR2 DEFAULT NULL));
```

Parameters

Table 42-142 INSERT_AUTOBIN_NUM_EQWIDTH Procedure Parameters

Parameter	Description

`bin_table_name`	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`VAL NUMBER`
	`BIN VARCHAR2(4000)`
	`CREATE_BIN_NUM` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_AUTOBIN_NUM_EQWIDTH`.

`data_table_name`	Name of the table containing the data to be transformed
`bin_num`	Minimum number of bins. If *`bin_num`* is 0 or `NULL`, it is ignored. The default value of *`bin_num`* is 3.
`max_bin_num`	Maximum number of bins. If *`max_bin_num`* is 0 or `NULL`, it is ignored. The default value of *`max_bin_num`* is 100.
`exclude_list`	List of numerical columns to be excluded from the binning process. If you do not specify *`exclude_list`*, all numerical columns in the data source are binned. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`round_num`	Specifies how to round the number in the `VAL` column of the transformation definition table. When *`round_num`* is positive, it specifies the most significant digits to retain. When *`round_num`* is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example. The default value of *`round_num`* is 6.
`sample_size`	Size of the data sample. If *`sample_size`* is less than the total number of non-`NULL` values in the column, then *`sample_size`* is used instead of the SQL `COUNT` function in computing the number of bins. If *`sample_size`* is 0 or `NULL`, it is ignored. See the Usage Notes. The default value of *`sample_size`* is 50,000.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.

Table 42-142 (Cont.) INSERT_AUTOBIN_NUM_EQWIDTH Procedure Parameters

Parameter	Description
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`rem_table_name`	Name of a transformation definition table for column removal. The table must have the columns described in "CREATE_COL_REM Procedure". `INSERT_AUTOBIN_NUM_EQWIDTH` ignores columns with all nulls or only one unique value. If you specify a value for *`rem_table_name`*, these columns are removed from the mining data. If you do not specify a value for *`rem_table_name`*, these unbinned columns remain in the data.
`rem_schema_name`	Schema of *`rem_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. `INSERT_AUTOBIN_NUM_EQWIDTH` computes the number of bins for a column based on the number of non-null values (`COUNT`), the maximum (`MAX`), the minimum (`MIN`), the standard deviation (`STDDEV`), and the constant C=3.49/0.9:

   ```
   N=floor(power(COUNT,1/3)*(max-min)/(c*dev))
   ```

   If the *`sample_size`* parameter is specified, it is used instead of `COUNT`.

   See *Oracle Machine Learning for SQL User's Guide* for information about the `COUNT`, `MAX`, `MIN`, `STDDEV`, `FLOOR`, and `POWER` functions.
3. `INSERT_AUTOBIN_NUM_EQWIDTH` uses absolute values to compute the number of bins. The sign of the parameters *`bin_num`*, *`max_bin_num`*, and *`sample_size`* has no effect on the result.
4. In computing the number of bins, `INSERT_AUTOBIN_NUM_EQWIDTH` evaluates the following criteria in the following order:
   a. The minimum number of bins (*`bin_num`*)
   b. The maximum number of bins (*`max_bin_num`*)
   c. The maximum number of bins for integer columns, calculated as the number of distinct values in the range *`max-min+1`*.
5. The *`round_num`* parameter controls the rounding of column values in the transformation definition table, as follows:

   ```
   For a value of 308.162:
   when round_num =  1       result is 300
   when round_num =  2       result is 310
   when round_num =  3       result is 308
   when round_num =  0       result is 308.162
   when round_num = -1       result is 308.16
   when round_num = -2       result is 308.2
   ```

Examples

In this example, `INSERT_AUTOBIN_NUM_EQWIDTH` computes the bin boundaries for the `cust_year_of_birth` column in `sh.customers` and inserts the transformations in a transformation definition table. The STACK_BIN_NUM Procedure creates a transformation list from the contents of the definition table. The CREATE_MODEL Procedure embeds the transformation list in a new model called `nb_model`.

The transformation and reverse transformation expressions embedded in `nb_model` are returned by the GET_MODEL_TRANSFORMATIONS Function.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_year_of_birth, cust_postal_code
       FROM sh.customers;

DESCRIBE mining_data
 Name                          Null?    Type
 ----------------------------- -------- ----------------------------
 CUST_ID                       NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH            NOT NULL NUMBER(4)
 CUST_POSTAL_CODE              NOT NULL VARCHAR2(10)

BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM(
     bin_table_name   => 'bin_tbl');
  dbms_data_mining_transform.INSERT_AUTOBIN_NUM_EQWIDTH (
     bin_table_name   => 'bin_tbl',
     data_table_name  => 'mining_data',
     bin_num          => 3,
     max_bin_num      => 5,
     exclude_list     => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/

set numwidth 4
column val off
SELECT col, val, bin FROM bin_tbl
      ORDER BY val ASC;

COL                          VAL BIN
------------------------- ---- -----
CUST_YEAR_OF_BIRTH        1913
CUST_YEAR_OF_BIRTH        1928 1
CUST_YEAR_OF_BIRTH        1944 2
CUST_YEAR_OF_BIRTH        1959 3
CUST_YEAR_OF_BIRTH        1975 4
CUST_YEAR_OF_BIRTH        1990 5

DECLARE
     year_birth_xform   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.STACK_BIN_NUM (
         bin_table_name           => 'bin_tbl',
         xform_list               =>  year_birth_xform);
     dbms_data_mining.CREATE_MODEL(
         model_name              => 'nb_model',
         mining_function         => dbms_data_mining.classification,
         data_table_name         => 'mining_data',
         case_id_column_name     => 'cust_id',
         target_column_name      => 'cust_postal_code',
```

```
          settings_table_name      => null,
          data_schema_name         => null,
          settings_schema_name     => null,
          xform_list               => year_birth_xform);
END;
/

SELECT attribute_name
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

ATTRIBUTE_NAME
------------------------
CUST_YEAR_OF_BIRTH

SELECT expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

EXPRESSION
--------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1913 THEN NULL WHEN "CUST_YEAR_OF_BIRTH"<=1928.4
 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1943.8 THEN '2' WHEN "CUST_YEAR_OF_BIRTH"
<=1959.2 THEN '3' WHEN "CUST_YEAR_OF_BIRTH"<=1974.6 THEN '4' WHEN
"CUST_YEAR_OF_BIRTH" <=1990 THEN '5' END

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

REVERSE_EXPRESSION
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",'5','(1974.6; 1990]','1','[1913; 1928.4]','2','(1928
.4; 1943.8]','3','(1943.8; 1959.2]','4','(1959.2; 1974.6]',NULL,'( ; 1913), (199
0;  ), NULL')
```

42.2.3.11 INSERT_BIN_CAT_FREQ Procedure

This procedure performs categorical binning and inserts the transformation definitions in a transformation definition table. The procedure computes the bin boundaries based on frequency.

`INSERT_BIN_CAT_FREQ` bins all the `CHAR` and `VARCHAR2` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_CAT_FREQ (
     bin_table_name        IN VARCHAR2,
     data_table_name       IN VARCHAR2,
     bin_num               IN PLS_INTEGER DEFAULT 9,
     exclude_list          IN COLUMN_LIST DEFAULT NULL,
     default_num           IN PLS_INTEGER DEFAULT 2,
     bin_support           IN NUMBER DEFAULT NULL,
     bin_schema_name       IN VARCHAR2 DEFAULT NULL,
     data_schema_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-143 INSERT_BIN_CAT_FREQ Procedure Parameters

Parameter	Description

`bin_table_name`	Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table.The following columns are required:
	`COL VARCHAR2(30)`
	`VAL VARCHAR2(4000)`
	`BIN VARCHAR2(4000)`
	`CREATE_BIN_CAT` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_BIN_CAT_FREQ`.

`data_table_name`	Name of the table containing the data to be transformed
`bin_num`	The number of bins to fill using frequency-based binning The total number of bins will be *`bin_num`*+1. The additional bin is the default bin. Classes that are not assigned to a frequency-based bin will be assigned to the default bin. The default binning order is from highest to lowest: the most frequently occurring class is assigned to the first bin, the second most frequently occurring class is assigned to the second bin, and so on.You can reverse the binning order by specifying a negative number for *`bin_num`*. The negative sign causes the binning order to be from lowest to highest. If the total number of distinct values (classes) in the column is less than *`bin_num`*, then a separate bin will be created for each value and the default bin will be empty. If you specify `NULL` or `0` for *`bin_num`*, no binning is performed. The default value of *`bin_num`* is 9.
`exclude_list`	List of categorical columns to be excluded from the binning process. If you do not specify *`exclude_list`*, all categorical columns in the data source are binned. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`default_num`	The number of class occurrences (rows of the same class) required for assignment to the default bin By default, *`default_num`* is the minimum number of occurrences required for assignment to the default bin. For example, if *`default_num`* is 3 and a given class occurs only once, it will not be assigned to the default bin. You can change the occurrence requirement from minimum to maximum by specifying a negative number for *`default_num`*. For example, if *`default_num`* is -3 and a given class occurs only once, it *will* be assigned to the default bin, but a class that occurs four or more times will not be included. If you specify `NULL` or `0` for *`default_bin`*, there are no requirements for assignment to the default bin. The default value of *`default_num`* is 2.

Table 42-143 (Cont.) INSERT_BIN_CAT_FREQ Procedure Parameters

Parameter	Description
`bin_support`	The number of class occurrences (rows of the same class) required for assignment to a frequency-based bin. *`bin_support`* is expressed as a fraction of the total number of rows. By default, *`bin_support`* is the minimum percentage required for assignment to a frequency-based bin. For example, if there are twenty rows of data and you specify.2 for *`bin_support`*, then there must be four or more occurrences of a class (.2*20) in order for it to be assigned to a frequency-based bin. You can change *`bin_support`* from a minimum percentage to a maximum percentage by specifying a negative number for *`bin_support`*. For example, if there are twenty rows of data and you specify -.2 for *`bin_support`*, then there must be four or less occurrences of a class in order for it to be assigned to a frequency-based bin. Classes that occur less than a positive *`bin_support`* or more than a negative *`bin_support`* will be assigned to the default bin. If you specify `NULL` or `0` for *`bin_support`*, then there is no support requirement for frequency-based binning. The default value of *`bin_support`* is `NULL`.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about categorical data.
2. If values occur with the same frequency, `INSERT_BIN_CAT_FREQ` assigns them in descending order when binning is from most to least frequent, or in ascending order when binning is from least to most frequent.

Examples

1. In this example, `INSERT_BIN_CAT_FREQ` computes the bin boundaries for the `cust_postal_code` and `cust_city` columns in `sh.customers` and inserts the transformations in a transformation definition table. The STACK_BIN_CAT Procedure creates a transformation list from the contents of the definition table, and the CREATE_MODEL Procedure embeds the transformation list in a new model called `nb_model`.

   The transformation and reverse transformation expressions embedded in `nb_model` are returned by the GET_MODEL_TRANSFORMATIONS Function.

```
CREATE OR REPLACE VIEW mining_data AS
          SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_city
          FROM sh.customers;

DESCRIBE mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
```

```
 CUST_POSTAL_CODE                      NOT NULL VARCHAR2(10)
 CUST_CITY                             NOT NULL VARCHAR2(30)

BEGIN
    dbms_data_mining_transform.CREATE_BIN_CAT(
       bin_table_name   => 'bin_tbl_1');
    dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
       bin_table_name   => 'bin_tbl_1',
       data_table_name  => 'mining_data',
       bin_num          => 4);
END;
/

column col format a18
column val format a15
column bin format a10
SELECT col, val, bin
       FROM bin_tbl_1
       ORDER BY col ASC, bin ASC;

COL                VAL             BIN
------------------ --------------- ----------
CUST_CITY          Los Angeles     1
CUST_CITY          Greenwich       2
CUST_CITY          Killarney       3
CUST_CITY          Montara         4
CUST_CITY                          5
CUST_POSTAL_CODE   38082           1
CUST_POSTAL_CODE   63736           2
CUST_POSTAL_CODE   55787           3
CUST_POSTAL_CODE   78558           4
CUST_POSTAL_CODE                   5

DECLARE
      city_xform   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
      dbms_data_mining_transform.STACK_BIN_CAT (
          bin_table_name           => 'bin_tbl_1',
          xform_list               =>  city_xform);
      dbms_data_mining.CREATE_MODEL(
           model_name              => 'nb_model',
           mining_function         => dbms_data_mining.classification,
           data_table_name         => 'mining_data',
           case_id_column_name     => 'cust_id',
           target_column_name      => 'cust_city',
           settings_table_name     => null,
           data_schema_name        => null,
           settings_schema_name    => null,
           xform_list              => city_xform);
END;
/

SELECT attribute_name
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

ATTRIBUTE_NAME
-----------------------------------------------------------------------------
CUST_CITY
CUST_POSTAL_CODE

SELECT expression
```

```
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

EXPRESSION
-----------------------------------------------------------------------------
DECODE("CUST_CITY",'Greenwich','2','Killarney','3','Los Angeles','1',
'Montara','4',NULL,NULL,'5')
DECODE("CUST_POSTAL_CODE",'38082','1','55787','3','63736','2','78558','4',NULL,NULL,'5')

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

REVERSE_EXPRESSION
-----------------------------------------------------------------------------
DECODE("CUST_CITY",'2','''Greenwich''','3','''Killarney''','1',
'''Los Angeles''','4','''Montara''',NULL,'NULL','5','DEFAULT')
DECODE("CUST_POSTAL_CODE",'1','''38082''','3','''55787''','2','''63736''',
'4','''78558''',NULL,'NULL','5','DEFAULT')
```

2. The binning order in example 1 is from most frequent to least frequent. The following example shows reverse order binning (least frequent to most frequent). The binning order is reversed by setting *bin_num* to -4 instead of 4.

```
BEGIN
    dbms_data_mining_transform.CREATE_BIN_CAT(
        bin_table_name   => 'bin_tbl_reverse');
    dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
        bin_table_name   => 'bin_tbl_reverse',
        data_table_name  => 'mining_data',
        bin_num          => -4);
 END;
 /

column col format a20
SELECT col, val, bin
       FROM bin_tbl_reverse
       ORDER BY col ASC, bin ASC;

COL                  VAL             BIN
-------------------- --------------- ----------
CUST_CITY            Tokyo           1
CUST_CITY            Sliedrecht      2
CUST_CITY            Haarlem         3
CUST_CITY            Diemen          4
CUST_CITY                            5
CUST_POSTAL_CODE     49358           1
CUST_POSTAL_CODE     80563           2
CUST_POSTAL_CODE     74903           3
CUST_POSTAL_CODE     71349           4
CUST_POSTAL_CODE                     5
```

42.2.3.12 INSERT_BIN_NUM_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

`INSERT_BIN_NUM_EQWIDTH` computes a specified number of bins (*n*) and assigns *(max-min)/n* values to each bin. The number of bins is the same for each column. If you want to use equi-width binning, but you want the number of bins to be calculated on a per-column basis, use the INSERT_AUTOBIN_NUM_EQWIDTH Procedure.

INSERT_BIN_NUM_EQWIDTH bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_EQWIDTH (
     bin_table_name         IN VARCHAR2,
     data_table_name        IN VARCHAR2,
     bin_num                IN PLS_INTEGER DEFAULT 10,
     exclude_list           IN COLUMN_LIST DEFAULT NULL,
     round_num              IN PLS_INTEGER DEFAULT 6,
     bin_schema_name        IN VARCHAR2 DEFAULT NULL,
     data_schema_name       IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-144 INSERT_BIN_NUM_EQWIDTH Procedure Parameters

Parameter	Description

bin_table_name	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`VAL NUMBER`
	`BIN VARCHAR2(4000)`
	CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_NUM_EQWIDTH.

data_table_name	Name of the table containing the data to be transformed
bin_num	Number of bins. No binning occurs if *bin_num* is 0 or NULL. The default number of bins is 10.
exclude_list	List of numerical columns to be excluded from the binning process. If you do not specify *exclude_list*, all numerical columns in the data source are binned. The format of *exclude_list* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
round_num	Specifies how to round the number in the VAL column of the transformation definition table. When *round_num* is positive, it specifies the most significant digits to retain. When *round_num* is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example. The default value of *round_num* is 6.
bin_schema_name	Schema of *bin_table_name*. If no schema is specified, the current schema is used.
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. The *round_num* parameter controls the rounding of column values in the transformation definition table, as follows:

   ```
   For a value of 308.162:
   when round_num =  1       result is 300
   when round_num =  2       result is 310
   when round_num =  3       result is 308
   when round_num =  0       result is 308.162
   when round_num = -1       result is 308.16
   when round_num = -2       result is 308.2
   ```

3. `INSERT_BIN_NUM_EQWIDTH` ignores columns with all `NULL` values or only one unique value.

Examples

In this example, `INSERT_BIN_NUM_EQWIDTH` computes the bin boundaries for the `affinity_card` column in `mining_data_build` and inserts the transformations in a transformation definition table. The STACK_BIN_NUM Procedure creates a transformation list from the contents of the definition table. The CREATE_MODEL Procedure embeds the transformation list in a new model called `glm_model`.

The transformation and reverse transformation expressions embedded in `glm_model` are returned by the GET_MODEL_TRANSFORMATIONS Function.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_income_level, cust_gender, affinity_card
       FROM mining_data_build;

DESCRIBE mining_data
 Name                          Null?    Type
 ------------------------- -------- -----------------
 CUST_ID                       NOT NULL NUMBER
 CUST_INCOME_LEVEL                      VARCHAR2(30)
 CUST_GENDER                            VARCHAR2(1)
 AFFINITY_CARD                          NUMBER(10)

BEGIN
    dbms_data_mining_transform.CREATE_BIN_NUM(
        bin_table_name   => 'bin_tbl');
    dbms_data_mining_transform.INSERT_BIN_NUM_EQWIDTH (
        bin_table_name   => 'bin_tbl',
        data_table_name  => 'mining_data',
        bin_num          => 4,
        exclude_list     => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/

set numwidth 10
column val off
column col format a20
column bin format a10
SELECT col, val, bin FROM bin_tbl
    ORDER BY val ASC;
```

```
COL                         VAL  BIN
-------------------- ----------  ----------
AFFINITY_CARD                 0
AFFINITY_CARD               .25  1
AFFINITY_CARD                .5  2
AFFINITY_CARD               .75  3
AFFINITY_CARD                 1  4

CREATE TABLE glmsettings(
        setting_name  VARCHAR2(30),
        setting_value VARCHAR2(30));

BEGIN
   INSERT INTO glmsettings (setting_name, setting_value) VALUES
         (dbms_data_mining.algo_name, dbms_data_mining.algo_generalized_linear_model);
   COMMIT;
END;
/

DECLARE
     xforms   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.STACK_BIN_NUM (
         bin_table_name           => 'bin_tbl',
         xform_list               =>  xforms,
         literal_flag             =>  TRUE);
     dbms_data_mining.CREATE_MODEL(
         model_name              => 'glm_model',
         mining_function         => dbms_data_mining.regression,
         data_table_name         => 'mining_data',
         case_id_column_name     => 'cust_id',
         target_column_name      => 'affinity_card',
         settings_table_name     => 'glmsettings',
         data_schema_name        => null,
         settings_schema_name    => null,
         xform_list              => xforms);
END;
/

SELECT attribute_name
      FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

ATTRIBUTE_NAME
------------------------
AFFINITY_CARD

SELECT expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

EXPRESSION
--------------------------------------------------------------------------------
CASE WHEN "AFFINITY_CARD"<0 THEN NULL WHEN "AFFINITY_CARD"<=.25 THEN 1 WHEN
"AFFINITY_CARD"<=.5 THEN 2 WHEN "AFFINITY_CARD"<=.75 THEN 3 WHEN
"AFFINITY_CARD"<=1 THEN 4 END

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

REVERSE_EXPRESSION
--------------------------------------------------------------------------------
```

```
DECODE("AFFINITY_CARD",4,'(.75; 1]',1,'[0; .25]',2,'(.25; .5]',3,'(.5; .75]',
NULL,'( ; 0), (1;  ), NULL')
```

42.2.3.13 INSERT_BIN_NUM_QTILE Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure calls the SQL `NTILE` function to order the data and divide it equally into the specified number of bins (quantiles).

`INSERT_BIN_NUM_QTILE` bins all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_QTILE (
    bin_table_name       IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    bin_num              IN PLS_INTEGER DEFAULT 10,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    bin_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-145 INSERT_BIN_NUM_QTILE Procedure Parameters

Parameter	Description

`bin_table_name`	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:
	`COL        VARCHAR2(30)`
	`VAL        NUMBER`
	`BIN        VARCHAR2(4000)`
	`CREATE_BIN_NUM` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_BIN_NUM_QTILE`.

`data_table_name`	Name of the table containing the data to be transformed
`bin_num`	Number of bins. No binning occurs if *`bin_num`* is `0` or `NULL`. The default number of bins is 10.
`exclude_list`	List of numerical columns to be excluded from the binning process. If you do not specify *`exclude_list`*, all numerical columns in the data source are binned. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. After dividing the data into quantiles, the `NTILE` function distributes any remainder values one for each quantile, starting with the first. See *Oracle Database SQL Language Reference* for details.
3. Columns with all `NULL` values are ignored by `INSERT_BIN_NUM_QTILE`.

Examples

In this example, `INSERT_BIN_NUM_QTILE` computes the bin boundaries for the `cust_year_of_birth` and `cust_credit_limit` columns in `sh.customers` and inserts the transformations in a transformation definition table. The STACK_BIN_NUM Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in `STACK_VIEW`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
        SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
        FROM sh.customers;

DESCRIBE mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_CREDIT_LIMIT                         NUMBER
 CUST_CITY                                 NOT NULL VARCHAR2(30)

BEGIN
   dbms_data_mining_transform.CREATE_BIN_NUM(
        bin_table_name   => 'bin_tbl');
   dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (
        bin_table_name   => 'bin_tbl',
        data_table_name  => 'mining_data',
        bin_num          => 3,
        exclude_list     => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/

set numwidth 8
column val off
column col format a20
column bin format a10
SELECT col, val, bin
      FROM bin_tbl
      ORDER BY col ASC, val ASC;

COL                       VAL BIN
-------------------- -------- ----------
CUST_CREDIT_LIMIT        1500
CUST_CREDIT_LIMIT        3000 1
CUST_CREDIT_LIMIT        9000 2
CUST_CREDIT_LIMIT       15000 3
CUST_YEAR_OF_BIRTH       1913
CUST_YEAR_OF_BIRTH       1949 1
CUST_YEAR_OF_BIRTH       1965 2
```

```
CUST_YEAR_OF_BIRTH       1990 3

DECLARE
   xforms   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
   dbms_data_mining_transform.STACK_BIN_NUM (
        bin_table_name        => 'bin_tbl',
        xform_list            =>  xforms);
   dbms_data_mining_transform.XFORM_STACK (
        xform_list            =>  xforms,
        data_table_name       => 'mining_data',
        xform_view_name       => 'stack_view');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name in 'STACK_VIEW';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH"<1913 THEN NULL WHEN "CUST_YEAR_O
F_BIRTH"<=1949 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1965 THEN '2' WHEN "CUST_YEAR
_OF_BIRTH"<=1990 THEN '3' END "CUST_YEAR_OF_BIRTH",CASE WHEN "CUST_CREDIT_LIMIT"
<1500 THEN NULL WHEN "CUST_CREDIT_LIMIT"<=3000 THEN '1' WHEN "CUST_CREDIT_LIMIT"
<=9000 THEN '2' WHEN "CUST_CREDIT_LIMIT"<=15000 THEN '3' END "CUST_CREDIT_LIMIT"
,"CUST_CITY" FROM mining_data
```

42.2.3.14 INSERT_BIN_SUPER Procedure

This procedure performs numerical and categorical binning and inserts the transformation definitions in transformation definition tables. The procedure computes bin boundaries based on intrinsic relationships between predictors and a target.

`INSERT_BIN_SUPER` uses an intelligent binning technique known as **supervised binning**. It builds a single-predictor decision tree and derives the bin boundaries from splits within the tree.

`INSERT_BIN_SUPER` bins all the `VARCHAR2`, `CHAR`, `NUMBER`, and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_SUPER (
    num_table_name        IN VARCHAR2,
    cat_table_name        IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    target_column_name    IN VARCHAR2,
    max_bin_num           IN PLS_INTEGER  DEFAULT 1000,
    exclude_list          IN COLUMN_LIST  DEFAULT NULL,
    num_schema_name       IN VARCHAR2     DEFAULT NULL,
    cat_schema_name       IN VARCHAR2     DEFAULT NULL,
    data_schema_name      IN VARCHAR2     DEFAULT NULL,
    rem_table_name        IN VARCHAR2     DEFAULT NULL,
    rem_schema_name       IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table 42-146 INSERT_BIN_SUPER Procedure Parameters

Parameter	Description

`num_table_name`	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`VAL VNUMBER`
	`BIN VARCHAR2(4000)`
	`CREATE_BIN_NUM` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_BIN_SUPER`.

`cat_table_name`	Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`VAL VARCHAR2(4000)`
	`BIN VARCHAR2(4000)`
	`CREATE_BIN_CAT` creates an additional column, `ATT`, which is used for specifying nested attributes. This column is not used by `INSERT_BIN_SUPER`.

`data_table_name`	Name of the table containing the data to be transformed
`target_column_name`	Name of a column to be used as the target for the decision tree models
`max_bin_num`	The maximum number of bins. The default is 1000.
`exclude_list`	List of columns to be excluded from the binning process. If you do not specify *`exclude_list`*, all numerical and categorical columns in the data source are binned. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`num_schema_name`	Schema of *`num_table_name`*. If no schema is specified, the current schema is used.
`cat_schema_name`	Schema of *`cat_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`rem_table_name`	Name of a column removal definition table. The table must have the columns described in "CREATE_COL_REM Procedure". You can use `CREATE_COL_REM` to create the table. See Usage Notes.
`rem_schema_name`	Schema of *`rem_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical and categorical data.
2. Columns that have no significant splits are not binned. You can remove the unbinned columns from the mining data by specifying a column removal definition table. If you do not specify a column removal definition table, the unbinned columns remain in the mining data.
3. See *Oracle Machine Learning for SQL Concepts* to learn more about decision trees in Oracle Machine Learning for SQL

Examples

In this example, `INSERT_BIN_SUPER` computes the bin boundaries for predictors of `cust_credit_limit` and inserts the transformations in transformation definition tables. One predictor is numerical, the other is categorical. (`INSERT_BIN_SUPER` determines that the `cust_postal_code` column is not a significant predictor.) `STACK` procedures create transformation lists from the contents of the definition tables.

The SQL expressions that compute the transformations are shown in the views `MINING_DATA_STACK_NUM` and `MINING_DATA_STACK_CAT`. The views are for display purposes only; they cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
     SELECT cust_id, cust_year_of_birth, cust_marital_status,
            cust_postal_code, cust_credit_limit
     FROM sh.customers;

DESCRIBE mining_data
 Name                              Null?    Type
 --------------------------------- -------- ------------------------------------
 CUST_ID                           NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                NOT NULL NUMBER(4)
 CUST_MARITAL_STATUS                        VARCHAR2(20)
 CUST_POSTAL_CODE                  NOT NULL VARCHAR2(10)
 CUST_CREDIT_LIMIT                          NUMBER

BEGIN
    dbms_data_mining_transform.CREATE_BIN_NUM(
         bin_table_name      => 'bin_num_tbl');
    dbms_data_mining_transform.CREATE_BIN_CAT(
         bin_table_name      => 'bin_cat_tbl');
    dbms_data_mining_transform.CREATE_COL_REM(
         rem_table_name      => 'rem_tbl');
END;
/

BEGIN
   COMMIT;
   dbms_data_mining_transform.INSERT_BIN_SUPER (
      num_table_name      => 'bin_num_tbl',
      cat_table_name      => 'bin_cat_tbl',
      data_table_name     => 'mining_data',
      target_column_name  => 'cust_credit_limit',
      max_bin_num         =>  4,
      exclude_list        =>  dbms_data_mining_transform.COLUMN_LIST('cust_id'),
      num_schema_name     => 'oml_user',
      cat_schema_name     => 'oml_user',
```

```
      data_schema_name    => 'oml_user',
      rem_table_name      => 'rem_tbl',
      rem_schema_name     => 'oml_user');
   COMMIT;
END;
/

set numwidth 8
column val off
SELECT col, val, bin FROM bin_num_tbl
      ORDER BY bin ASC;

COL                       VAL BIN
-------------------- -------- ----------
CUST_YEAR_OF_BIRTH     1923.5 1
CUST_YEAR_OF_BIRTH     1923.5 1
CUST_YEAR_OF_BIRTH     1945.5 2
CUST_YEAR_OF_BIRTH     1980.5 3
CUST_YEAR_OF_BIRTH            4

column val on
column val format a20
SELECT col, val, bin FROM bin_cat_tbl
      ORDER BY bin ASC;

COL                  VAL                  BIN
-------------------- -------------------- ----------
CUST_MARITAL_STATUS  married              1
CUST_MARITAL_STATUS  single               2
CUST_MARITAL_STATUS  Mar-AF               3
CUST_MARITAL_STATUS  Mabsent              3
CUST_MARITAL_STATUS  Divorc.              3
CUST_MARITAL_STATUS  Married              3
CUST_MARITAL_STATUS  Widowed              3
CUST_MARITAL_STATUS  NeverM               3
CUST_MARITAL_STATUS  Separ.               3
CUST_MARITAL_STATUS  divorced             4
CUST_MARITAL_STATUS  widow                4

SELECT col from rem_tbl;

COL
--------------------
CUST_POSTAL_CODE

DECLARE
    xforms_num      dbms_data_mining_transform.TRANSFORM_LIST;
    xforms_cat      dbms_data_mining_transform.TRANSFORM_LIST;
    BEGIN
       dbms_data_mining_transform.STACK_BIN_NUM (
            bin_table_name     => 'bin_num_tbl',
            xform_list         => xforms_num);
       dbms_data_mining_transform.XFORM_STACK (
            xform_list          => xforms_num,
            data_table_name     => 'mining_data',
            xform_view_name     => 'mining_data_stack_num');
       dbms_data_mining_transform.STACK_BIN_CAT (
             bin_table_name     => 'bin_cat_tbl',
             xform_list         => xforms_cat);
       dbms_data_mining_transform.XFORM_STACK (
             xform_list          => xforms_cat,
```

```
            data_table_name    => 'mining_data',
            xform_view_name    => 'mining_data_stack_cat');
   END;
 /

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK_NUM';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH"<1923.5 THEN '1' WHEN "CUST_YEAR_
OF_BIRTH"<=1923.5 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1945.5 THEN '2' WHEN "CUST
_YEAR_OF_BIRTH"<=1980.5 THEN '3' WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN '4'
END "CUST_YEAR_OF_BIRTH","CUST_MARITAL_STATUS","CUST_POSTAL_CODE","CUST_CREDIT_L
IMIT" FROM mining_data


SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK_CAT';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_YEAR_OF_BIRTH",DECODE("CUST_MARITAL_STATUS",'Divorc.','3'
,'Mabsent','3','Mar-AF','3','Married','3','NeverM','3','Separ.','3','Widowed','3
','divorced','4','married','1','single','2','widow','4') "CUST_MARITAL_STATUS","
CUST_POSTAL_CODE","CUST_CREDIT_LIMIT" FROM mining_data
```

42.2.3.15 INSERT_CLIP_TRIM_TAIL Procedure

This procedure replaces numeric outliers with nulls and inserts the transformation definitions in a transformation definition table.

`INSERT_CLIP_TRIM_TAIL` computes the boundaries of the data based on a specified percentage. It removes the values that fall outside the boundaries (tail values) from the data. If you wish to replace the tail values instead of removing them, use the INSERT_CLIP_WINSOR_TAIL Procedure.

`INSERT_CLIP_TRIM_TAIL` clips all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_TRIM_TAIL (
    clip_table_name      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    tail_frac            IN NUMBER DEFAULT 0.025,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    clip_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-147 INSERT_CLIP_TRIM_TAIL Procedure Parameters

Parameter	Description

`clip_table_name`	Name of the transformation definition table for numerical clipping. You can use the CREATE_CLIP Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`LCUT NUMBER`
	`LVAL NUMBER`
	`RCUT NUMBER`
	`RVAL NUMBER`
	`CREATE_CLIP` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_CLIP_TRIM_TAIL`.

`data_table_name`	Name of the table containing the data to be transformed
`tail_frac`	The percentage of non-null values to be designated as outliers at each end of the data. For example, if *`tail_frac`* is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers. If *`tail_frac`* is greater than or equal to .5, no clipping occurs. The default value of *`tail_frac`* is 0.025.
`exclude_list`	List of numerical columns to be excluded from the clipping process. If you do not specify *`exclude_list`*, all numerical columns in the data are clipped. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`clip_schema_name`	Schema of *`clip_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. The `DBMS_DATA_MINING_TRANSFORM` package provides two clipping procedures: `INSERT_CLIP_TRIM_TAIL` and `INSERT_CLIP_WINSOR_TAIL`. Both procedures compute the boundaries as follows:
   - Count the number of non-null values, ***`n`***, and sort them in ascending order
   - Calculate the number of outliers, ***`t`***, as ***`n*tail_frac`***
   - Define the lower boundary ***`lcut`*** as the value at position ***`1+floor(t)`***
   - Define the upper boundary ***rcut*** as the value at position ***`n-floor(t)`***

     (The SQL `FLOOR` function returns the largest integer less than or equal to ***`t`***.)
   - All values that are <= ***`lcut`*** or => ***`rcut`*** are designated as outliers.

`INSERT_CLIP_TRIM_TAIL` replaces the outliers with nulls, effectively removing them from the data.

`INSERT_CLIP_WINSOR_TAIL` assigns ***lcut*** to the low outliers and ***rcut*** to the high outliers.

Examples

In this example, `INSERT_CLIP_TRIM_TAIL` trims 10% of the data in two columns (5% from the high end and 5% from the low end) and inserts the transformations in a transformation definition table. The STACK_CLIP Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the trimming is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
       FROM sh.customers;

DESCRIBE mining_data
 Name                           Null?    Type
 ------------------------------- -------- -------------------
 CUST_ID                        NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH             NOT NULL NUMBER(4)
 CUST_CREDIT_LIMIT                       NUMBER
 CUST_CITY                      NOT NULL VARCHAR2(30)

BEGIN
   dbms_data_mining_transform.CREATE_CLIP(
      clip_table_name    => 'clip_tbl');
   dbms_data_mining_transform.INSERT_CLIP_TRIM_TAIL(
      clip_table_name    => 'clip_tbl',
      data_table_name    => 'mining_data',
      tail_frac          => 0.05,
      exclude_list       => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

SELECT col, lcut, lval, rcut, rval
      FROM clip_tbl
      ORDER BY col ASC;

COL                        LCUT     LVAL     RCUT     RVAL
-------------------- -------- -------- -------- --------
CUST_CREDIT_LIMIT        1500             11000
CUST_YEAR_OF_BIRTH       1934              1982

DECLARE
     xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.STACK_CLIP (
          clip_table_name    => 'clip_tbl',
          xform_list         => xforms);
     dbms_data_mining_transform.XFORM_STACK (
          xform_list         => xforms,
          data_table_name    => 'mining_data',
          xform_view_name    => 'mining_data_stack');
 END;
 /
```

```
set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN NULL WHEN "CUST_YEAR
_OF_BIRTH" > 1982 THEN NULL ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH",C
ASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NULL WHEN "CUST_CREDIT_LIMIT" > 11000 T
HEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT","CUST_CITY" FROM minin
g_data
```

42.2.3.16 INSERT_CLIP_WINSOR_TAIL Procedure

This procedure replaces numeric outliers with the upper or lower boundary values. It inserts the transformation definitions in a transformation definition table.

`INSERT_CLIP_WINSOR_TAIL` computes the boundaries of the data based on a specified percentage. It replaces the values that fall outside the boundaries (tail values) with the related boundary value. If you wish to set tail values to null, use the INSERT_CLIP_TRIM_TAIL Procedure.

`INSERT_CLIP_WINSOR_TAIL` clips all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_WINSOR_TAIL (
    clip_table_name     IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    tail_frac           IN NUMBER DEFAULT 0.025,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    clip_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-148 INSERT_CLIP_WINSOR_TAIL Procedure Parameters

Parameter	Description

`clip_table_name`	Name of the transformation definition table for numerical clipping. You can use the CREATE_CLIP Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`LCUT NUMBER`
	`LVAL NUMBER`
	`RCUT NUMBER`
	`RVAL NUMBER`
	`CREATE_CLIP` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_CLIP_WINSOR_TAIL`.

`data_table_name`	Name of the table containing the data to be transformed

Table 42-148 (Cont.) INSERT_CLIP_WINSOR_TAIL Procedure Parameters

Parameter	Description
`tail_frac`	The percentage of non-null values to be designated as outliers at each end of the data. For example, if *`tail_frac`* is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers. If *`tail_frac`* is greater than or equal to .5, no clipping occurs. The default value of *`tail_frac`* is 0.025.
`exclude_list`	List of numerical columns to be excluded from the clipping process. If you do not specify *`exclude_list`*, all numerical columns in the data are clipped. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`clip_schema_name`	Schema of *`clip_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. The `DBMS_DATA_MINING_TRANSFORM` package provides two clipping procedures: `INSERT_CLIP_WINSOR_TAIL` and `INSERT_CLIP_TRIM_TAIL`. Both procedures compute the boundaries as follows:
   - Count the number of non-null values, ***`n`***, and sort them in ascending order
   - Calculate the number of outliers, ***`t`***, as ***`n*tail_frac`***
   - Define the lower boundary ***`lcut`*** as the value at position ***`1+floor(t)`***
   - Define the upper boundary ***rcut*** as the value at position ***`n-floor(t)`***

     (The SQL `FLOOR` function returns the largest integer less than or equal to ***`t`***.)
   - All values that are <= ***`lcut`*** or => ***`rcut`*** are designated as outliers.

   `INSERT_CLIP_WINSOR_TAIL` assigns ***`lcut`*** to the low outliers and ***`rcut`*** to the high outliers.

   `INSERT_CLIP_TRIM_TAIL` replaces the outliers with nulls, effectively removing them from the data.

Examples

In this example, `INSERT_CLIP_WINSOR_TAIL` winsorizes 10% of the data in two columns (5% from the high end, and 5% from the low end) and inserts the transformations in a transformation definition table. The STACK_CLIP Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
        SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
        FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_CREDIT_LIMIT                                  NUMBER
 CUST_CITY                                 NOT NULL VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.CREATE_CLIP(
      clip_table_name   => 'clip_tbl');
  dbms_data_mining_transform.INSERT_CLIP_WINSOR_TAIL(
      clip_table_name   => 'clip_tbl',
      data_table_name   => 'mining_data',
      tail_frac         => 0.05,
      exclude_list      => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

SELECT col, lcut, lval, rcut, rval FROM clip_tbl
   ORDER BY col ASC;
COL                                LCUT     LVAL     RCUT     RVAL
------------------------------ -------- -------- -------- --------
CUST_CREDIT_LIMIT                  1500     1500    11000    11000
CUST_YEAR_OF_BIRTH                 1934     1934     1982     1982

DECLARE
   xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
   dbms_data_mining_transform.STACK_CLIP (
   clip_table_name    => 'clip_tbl',
   xform_list        => xforms);
dbms_data_mining_transform.XFORM_STACK (
   xform_list         => xforms,
   data_table_name    => 'mining_data',
   xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SQL> SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN 1934 WHEN "CUST_YEAR
_OF_BIRTH" > 1982 THEN 1982 ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH",C
ASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN 1500 WHEN "CUST_CREDIT_LIMIT" > 11000 T
HEN 11000 ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT","CUST_CITY" FROM mini
ng_data
```

42.2.3.17 INSERT_MISS_CAT_MODE Procedure

This procedure replaces missing categorical values with the value that occurs most frequently in the column (the mode). It inserts the transformation definitions in a transformation definition table.

`INSERT_MISS_CAT_MODE` replaces missing values in all `VARCHAR2` and `CHAR` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (
    miss_table_name     IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    miss_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-149 INSERT_MISS_CAT_MODE Procedure Parameters

Parameter	Description

`miss_table_name`	Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`VAL VARCHAR2(4000)`
	`CREATE_MISS_CAT` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_MISS_CAT_MODE`.

`data_table_name`	Name of the table containing the data to be transformed
`exclude_list`	List of categorical columns to be excluded from missing value treatment. If you do not specify *`exclude_list`*, all categorical columns are transformed. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`miss_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about categorical data.
2. If you wish to replace categorical missing values with a value other than the mode, you can edit the transformation definition table.

See Also:

Oracle Machine Learning for SQL User's Guide for information about default missing value treatment in Oracle Machine Learning for SQL

Example

In this example, `INSERT_MISS_CAT_MODE` computes missing value treatment for `cust_city` and inserts the transformation in a transformation definition table. The STACK_MISS_CAT Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
         SELECT cust_id, cust_year_of_birth, cust_city
         FROM sh.customers;

describe mining_data
 Name                             Null?    Type
 -------------------------------- -------- ----------------
 CUST_ID                          NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH               NOT NULL NUMBER(4)
 CUST_CITY                        NOT NULL VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.create_miss_cat(
      miss_table_name   => 'missc_tbl');
  dbms_data_mining_transform.insert_miss_cat_mode(
      miss_table_name   => 'missc_tbl',
      data_table_name   => 'mining_data');
END;
/

SELECT stats_mode(cust_city) FROM mining_data;

STATS_MODE(CUST_CITY)
------------------------------
Los Angeles

SELECT col, val
    from missc_tbl;

COL                            VAL
------------------------------ ------------------------------
CUST_CITY                      Los Angeles

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_MISS_CAT (
        miss_table_name    => 'missc_tbl',
        xform_list        => xforms);
    dbms_data_mining_transform.XFORM_STACK (
         xform_list         => xforms,
         data_table_name    => 'mining_data',
         xform_view_name    => 'mining_data_stack');
```

```
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_YEAR_OF_BIRTH",NVL("CUST_CITY",'Los Angeles') "CUST_CITY"
 FROM mining_data
```

42.2.3.18 INSERT_MISS_NUM_MEAN Procedure

This procedure replaces missing numerical values with the average (the mean) and inserts the transformation definitions in a transformation definition table.

`INSERT_MISS_NUM_MEAN` replaces missing values in all `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
    miss_table_name     IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    round_num           IN PLS_INTEGER DEFAULT 6,
    miss_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-150 INSERT_MISS_NUM_MEAN Procedure Parameters

Parameter	Description

`miss_table_name`	Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table.
	The following columns are required by `INSERT_MISS_NUM_MEAN`:
	`COL            VARCHAR2(30)`
	`VAL            NUMBER`
	`CREATE_MISS_NUM` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_MISS_NUM_MEAN`.

`data_table_name`	Name of the table containing the data to be transformed
`exclude_list`	List of numerical columns to be excluded from missing value treatment. If you do not specify *`exclude_list`*, all numerical columns are transformed. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`round_num`	The number of significant digits to use for the mean. The default number is 6.

Table 42-150 (Cont.) INSERT_MISS_NUM_MEAN Procedure Parameters

Parameter	Description
miss_schema_name	Schema of *miss_table_name*. If no schema is specified, the current schema is used.
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.

Usage Notes

1. See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.
2. If you wish to replace numerical missing values with a value other than the mean, you can edit the transformation definition table.

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for information about default missing value treatment in Oracle Machine Learning for SQL

Example

In this example, INSERT_MISS_NUM_MEAN computes missing value treatment for cust_year_of_birth and inserts the transformation in a transformation definition table. The STACK_MISS_NUM Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_city
    FROM sh.customers;

DESCRIBE mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_CITY                                 NOT NULL VARCHAR2(30)

BEGIN
   dbms_data_mining_transform.create_miss_num(
       miss_table_name   => 'missn_tbl');
   dbms_data_mining_transform.insert_miss_num_mean(
       miss_table_name   => 'missn_tbl',
       data_table_name   => 'mining_data',
       exclude_list      => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

set numwidth 4
column val off
SELECT col, val
```

```
  FROM missn_tbl;

COL                   VAL
-------------------- ----
CUST_YEAR_OF_BIRTH   1957

SELECT avg(cust_year_of_birth) FROM mining_data;

AVG(CUST_YEAR_OF_BIRTH)
-----------------------
                   1957

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_MISS_NUM (
         miss_table_name    => 'missn_tbl',
         xform_list        => xforms);
    dbms_data_mining_transform.XFORM_STACK (
         xform_list         => xforms,
         data_table_name    => 'mining_data',
         xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID",NVL("CUST_YEAR_OF_BIRTH",1957.4) "CUST_YEAR_OF_BIRTH","CUST_CIT
Y" FROM mining_data
```

42.2.3.19 INSERT_NORM_LIN_MINMAX Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

`INSERT_NORM_LIN_MINMAX` computes the minimum and maximum values from the data and sets the value of *`shift`* and *`scale`* as follows:

```
shift = min
scale = max - min
```

Normalization is computed as:

```
x_new = (x_old - shift)/scale
```

`INSERT_NORM_LIN_MINMAX` rounds the value of *`scale`* to a specified number of significant digits before storing it in the transformation definition table.

`INSERT_NORM_LIN_MINMAX` normalizes all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_MINMAX (
     norm_table_name      IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     exclude_list         IN COLUMN_LIST DEFAULT NULL,
```

```
      round_num           IN PLS_INTEGER DEFAULT 6,
      norm_schema_name    IN VARCHAR2 DEFAULT NULL,
      data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-151 *INSERT_NORM_LIN_MINMAX Procedure Parameters*

Parameter	Description

`norm_table_name`	Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`SHIFT NUMBER`
	`SCALE NUMBER`
	`CREATE_NORM_LIN` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_NORM_LIN_MINMAX`.

`data_table_name`	Name of the table containing the data to be transformed
`exclude_list`	List of numerical columns to be excluded from normalization. If you do not specify *`exclude_list`*, all numerical columns are transformed. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`round_num`	The number of significant digits to use for the minimum and maximum. The default number is 6.
`norm_schema_name`	Schema of *`norm_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.

Examples

In this example, `INSERT_NORM_LIN_MINMAX` normalizes the `cust_year_of_birth` column and inserts the transformation in a transformation definition table. The STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
         SELECT cust_id, cust_gender, cust_year_of_birth
         FROM sh.customers;

describe mining_data
 Name                                    Null?    Type
 ------------------------------------ -------- ----------------
 CUST_ID                                 NOT NULL NUMBER
```

```
 CUST_GENDER                          NOT NULL CHAR(1)
 CUST_YEAR_OF_BIRTH                   NOT NULL NUMBER(4)

BEGIN
       dbms_data_mining_transform.CREATE_NORM_LIN(
         norm_table_name  => 'norm_tbl');
       dbms_data_mining_transform.INSERT_NORM_LIN_MINMAX(
         norm_table_name  => 'norm_tbl',
         data_table_name  => 'mining_data',
         exclude_list     => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
         round_num        => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

COL                                 SHIFT      SCALE
------------------------------ ---------- ----------
CUST_YEAR_OF_BIRTH                   1910         77

DECLARE
     xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.STACK_NORM_LIN (
          norm_table_name    => 'norm_tbl',
          xform_list        => xforms);
     dbms_data_mining_transform.XFORM_STACK (
          xform_list         => xforms,
          data_table_name    => 'mining_data',
          xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_GENDER",("CUST_YEAR_OF_BIRTH"-1910)/77 "CUST_YEAR_OF_BIRT
H" FROM mining_data
```

42.2.3.20 INSERT_NORM_LIN_SCALE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

`INSERT_NORM_LIN_SCALE` computes the minimum and maximum values from the data and sets the value of *`shift`* and *`scale`* as follows:

```
shift = 0
scale = max(abs(max), abs(min))
```

Normalization is computed as:

```
x_new = (x_old)/scale
```

`INSERT_NORM_LIN_SCALE` rounds the value of *`scale`* to a specified number of significant digits before storing it in the transformation definition table.

INSERT_NORM_LIN_SCALE normalizes all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_SCALE (
     norm_table_name      IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     exclude_list         IN COLUMN_LIST DEFAULT NULL,
     round_num            IN PLS_INTEGER DEFAULT 6,
     norm_schema_name     IN VARCHAR2 DEFAULT NULL,
     data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-152 INSERT_NORM_LIN_SCALE Procedure Parameters

Parameter	Description

norm_table_name	Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The following columns are required:
	`COL      VARCHAR2(30)`
	`SHIFT    NUMBER`
	`SCALE    NUMBER`
	CREATE_NORM_LIN creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_NORM_LIN_SCALE.

data_table_name	Name of the table containing the data to be transformed
exclude_list	List of numerical columns to be excluded from normalization. If you do not specify *exclude_list*, all numerical columns are transformed. The format of *exclude_list* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
round_num	The number of significant digits to use for *scale*. The default number is 6.
norm_schema_name	Schema of *norm_table_name*. If no schema is specified, the current schema is used.
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.

Usage Notes

See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.

Examples

In this example, INSERT_NORM_LIN_SCALE normalizes the cust_year_of_birth column and inserts the transformation in a transformation definition table. The STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING_DATA_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
      SELECT cust_id, cust_gender, cust_year_of_birth
      FROM sh.customers;

DESCRIBE mining_data
 Name                                  Null?    Type
 ---------------------------------- -------- ------------------
 CUST_ID                               NOT NULL NUMBER
 CUST_GENDER                           NOT NULL CHAR(1)
 CUST_YEAR_OF_BIRTH                    NOT NULL NUMBER(4)

BEGIN
   dbms_data_mining_transform.CREATE_NORM_LIN(
       norm_table_name  => 'norm_tbl');
       dbms_data_mining_transform.INSERT_NORM_LIN_SCALE(
       norm_table_name  => 'norm_tbl',
       data_table_name  => 'mining_data',
       exclude_list     => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
       round_num        => 3);
 END;
 /

SELECT col, shift, scale FROM norm_tbl;

COL                  SHIFT SCALE
-------------------- ----- -----
CUST_YEAR_OF_BIRTH       0  1990

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_NORM_LIN (
        norm_table_name    => 'norm_tbl',
        xform_list        => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list         => xforms,
        data_table_name    => 'mining_data',
        xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_GENDER",("CUST_YEAR_OF_BIRTH"-0)/1990 "CUST_YEAR_OF_BIRTH
" FROM mining_data
```

42.2.3.21 INSERT_NORM_LIN_ZSCORE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table.

`INSERT_NORM_LIN_ZSCORE` computes the mean and the standard deviation from the data and sets the value of *`shift`* and *`scale`* as follows:

```
shift = mean
scale = stddev
```

Normalization is computed as:

x_new = (*x_old* - *shift*)/*scale*

`INSERT_NORM_LIN_ZSCORE` rounds the value of *`scale`* to a specified number of significant digits before storing it in the transformation definition table.

`INSERT_NORM_LIN_ZSCORE` normalizes all the `NUMBER` and `FLOAT` columns in the data unless you specify a list of columns to ignore.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_ZSCORE (
     norm_table_name     IN VARCHAR2,
     data_table_name     IN VARCHAR2,
     exclude_list        IN COLUMN_LIST DEFAULT NULL,
     round_num           IN PLS_INTEGER DEFAULT 6,
     norm_schema_name    IN VARCHAR2 DEFAULT NULL,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-153 INSERT_NORM_LIN_ZSCORE Procedure Parameters

Parameter	Description

`norm_table_name`	Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The following columns are required:
	`COL VARCHAR2(30)`
	`SHIFT NUMBER`
	`SCALE NUMBER`
	`CREATE_NORM_LIN` creates an additional column, `ATT`, which may be used for specifying nested attributes. This column is not used by `INSERT_NORM_LIN_ZSCORE`.

`data_table_name`	Name of the table containing the data to be transformed
`exclude_list`	List of numerical columns to be excluded from normalization. If you do not specify *`exclude_list`*, all numerical columns are transformed. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`round_num`	The number of significant digits to use for *`scale`*. The default number is 6.
`norm_schema_name`	Schema of *`norm_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See *Oracle Machine Learning for SQL User's Guide* for details about numerical data.

Examples

In this example, `INSERT_NORM_LIN_ZSCORE` normalizes the `cust_year_of_birth` column and inserts the transformation in a transformation definition table. The STACK_NORM_LIN Procedure creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_gender, cust_year_of_birth
       FROM sh.customers;

DESCRIBE mining_data
 Name                                   Null?    Type
 -------------------------------------- -------- --------------------
 CUST_ID                                NOT NULL NUMBER
 CUST_GENDER                            NOT NULL CHAR(1)
 CUST_YEAR_OF_BIRTH                     NOT NULL NUMBER(4)

BEGIN
    dbms_data_mining_transform.CREATE_NORM_LIN(
       norm_table_name  => 'norm_tbl');
       dbms_data_mining_transform.INSERT_NORM_LIN_ZSCORE(
       norm_table_name  => 'norm_tbl',
       data_table_name  => 'mining_data',
       exclude_list     => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
       round_num        => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

COL                  SHIFT SCALE
-------------------- ----- -----
CUST_YEAR_OF_BIRTH    1960    15

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_NORM_LIN (
        norm_table_name    => 'norm_tbl',
        xform_list        => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list         => xforms,
        data_table_name    => 'mining_data',
        xform_view_name    => 'mining_data_stack');
END;
/

set long 3000
SQL> SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_GENDER",("CUST_YEAR_OF_BIRTH"-1960)/15 "CUST_YEAR_OF_BIRT
H" FROM mining_data
```

42.2.3.22 SET_EXPRESSION Procedure

This procedure appends a row to a VARCHAR2 array that stores a SQL expression.

The array can be used for specifying a transformation expression that is too long to be used with the SET_TRANSFORM Procedure.

The GET_EXPRESSION Function returns a row in the array.

When you use SET_EXPRESSION to build a transformation expression, you must build a corresponding reverse transformation expression, create a transformation record, and add the transformation record to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_EXPRESSION (
          expression    IN OUT NOCOPY EXPRESSION_REC,
          chunk                       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-154 SET_EXPRESSION Procedure Parameters

Parameter	Description
`expression`	An expression record (EXPRESSION_REC) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a VARCHAR2 array and index fields for specifying upper and lower boundaries within the array. There are two EXPRESSION_REC fields within a transformation record (TRANSFORM_REC): one for the transformation expression; the other for the reverse transformation expression. See Table 42-123 for a description of the EXPRESSION_REC type.
`chunk`	A VARCHAR2 chunk (row) to be appended to *expression*.

Notes

1. You can pass NULL in the *chunk* argument to SET_EXPRESSION to clear the previous chunk. The default value of *chunk* is NULL.
2. See "About Transformation Lists".
3. See "Operational Notes".

Examples

In this example, two calls to SET_EXPRESSION construct a transformation expression and two calls construct the reverse transformation.

> **Note:**
>
> This example is for illustration purposes only. It shows how `SET_EXPRESSION` appends the text provided in *`chunk`* to the text that already exists in *`expression`*. The `SET_EXPRESSION` procedure is meant for constructing very long transformation expressions that cannot be specified in a `VARCHAR2` argument to `SET_TRANSFORM`.
>
> Similarly while transformation lists are intended for embedding in a model, the transformation list `v_xlst` is shown in an external view for illustration purposes.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_credit_limit
       FROM sh.customers;

DECLARE
        v_expr dbms_data_mining_transform.EXPRESSION_REC;
        v_rexp dbms_data_mining_transform.EXPRESSION_REC;
        v_xrec dbms_data_mining_transform.TRANSFORM_REC;
        v_xlst dbms_data_mining_transform.TRANSFORM_LIST :=
                                dbms_data_mining_transform.TRANSFORM_LIST(NULL);
BEGIN
    dbms_data_mining_transform.SET_EXPRESSION(
         EXPRESSION  => v_expr,
         CHUNK       => '("CUST_YEAR_OF_BIRTH"-1910)');
    dbms_data_mining_transform.SET_EXPRESSION(
          EXPRESSION  => v_expr,
          CHUNK       => '/77');
    dbms_data_mining_transform.SET_EXPRESSION(
          EXPRESSION  => v_rexp,
          CHUNK       => '"CUST_YEAR_OF_BIRTH"*77');
    dbms_data_mining_transform.SET_EXPRESSION(
          EXPRESSION  => v_rexp,
          CHUNK       => '+1910');

    v_xrec := null;
    v_xrec.attribute_name := 'CUST_YEAR_OF_BIRTH';
    v_xrec.expression := v_expr;
    v_xrec.reverse_expression := v_rexp;
    v_xlst.TRIM;
    v_xlst.extend(1);
    v_xlst(1) := v_xrec;

    dbms_data_mining_transform.XFORM_STACK (
        xform_list          =>  v_xlst,
        data_table_name     => 'mining_data',
        xform_view_name     => 'v_xlst_view');

    dbms_output.put_line('====');
    FOR i IN 1..v_xlst.count LOOP
      dbms_output.put_line('ATTR: '||v_xlst(i).attribute_name);
      dbms_output.put_line('SUBN: '||v_xlst(i).attribute_subname);
      FOR j IN v_xlst(i).expression.lb..v_xlst(i).expression.ub LOOP
        dbms_output.put_line('EXPR: '||v_xlst(i).expression.lstmt(j));
      END LOOP;
      FOR j IN v_xlst(i).reverse_expression.lb..
```

```
                v_xlst(i).reverse_expression.ub LOOP
        dbms_output.put_line('REXP: '||v_xlst(i).reverse_expression.lstmt(j));
      END LOOP;
      dbms_output.put_line('====');
    END LOOP;
  END;
/
====
ATTR: CUST_YEAR_OF_BIRTH
SUBN:
EXPR: ("CUST_YEAR_OF_BIRTH"-1910)
EXPR: /77
REXP: "CUST_YEAR_OF_BIRTH"*77
REXP: +1910
====
```

42.2.3.23 SET_TRANSFORM Procedure

This procedure appends the transformation instructions for an attribute to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM (
          xform_list                IN OUT NOCOPY TRANSFORM_LIST,
          attribute_name            VARCHAR2,
          attribute_subname         VARCHAR2,
          expression                VARCHAR2,
          reverse_expression        VARCHAR2,
          attribute_spec            VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-155 SET_TRANSFORM Procedure Parameters

Parameter	Description
`xform_list`	A transformation list. See Table 42-123for a description of the `TRANSFORM_LIST` object type.
`attribute_name`	Name of the attribute to be transformed
`attribute_subname`	Name of the nested attribute if *`attribute_name`* is a nested column, otherwise `NULL`.
`expression`	A SQL expression that specifies the transformation of the attribute.
`reverse_expression`	A SQL expression that reverses the transformation for readability in model details and in the target of a supervised model (if the attribute is a target)

Table 42-155 (Cont.) SET_TRANSFORM Procedure Parameters

Parameter	Description
`attribute_spec`	One or more keywords that identify special treatment for the attribute during model build. Values are: • `NOPREP` — When ADP is on, prevents automatic transformation of the attribute. If ADP is not on, this value has no effect. • `TEXT` — Causes the attribute to be treated as unstructured text data • `FORCE_IN` — Forces the inclusion of the attribute in the model build. Applies only to GLM models with feature selection enabled (`ftr_selection_enable` = yes). Feature selection is disabled by default. If the model is not using GLM with feature selection, this value has no effect. See "Specifying Transformation Instructions for an Attribute" in *Oracle Machine Learning for SQL User's Guide*for more information about `attribute_spec`.

Usage Notes

1. See the following relevant sections in "Operational Notes":
   - About Transformation Lists
   - Nested Data Transformations
2. As shown in the following example, you can eliminate an attribute by specifying a null transformation expression and reverse expression. You can also use the STACK interface to remove a column (CREATE_COL_REM Procedure and STACK_COL_REM Procedure).

42.2.3.24 STACK_BIN_CAT Procedure

This procedure adds categorical binning transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_CAT (
     bin_table_name     IN            VARCHAR2,
     xform_list         IN OUT NOCOPY TRANSFORM_LIST,
     literal_flag       IN            BOOLEAN  DEFAULT FALSE,
     bin_schema_name    IN            VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-156 STACK_BIN_CAT Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_BIN_CAT`. To populate the table, you can use one of the `INSERT` procedures for categorical binning or you can write your own SQL. See Table 42-126
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`literal_flag`	Indicates whether the values in the `bin` column in the transformation definition table are valid SQL literals. When *`literal_flag`* is `FALSE` (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set *`literal_flag`* to `TRUE` if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See "INSERT_BIN_NUM_EQWIDTH Procedure" for an example.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how a binning transformation for the categorical column `cust_postal_code` could be added to a stack called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE or REPLACE VIEW mining_data AS
    SELECT cust_id, cust_postal_code, cust_credit_limit
       FROM sh.customers
       WHERE cust_id BETWEEN 100050 AND 100100;
BEGIN
```

```
   dbms_data_mining_transform.CREATE_BIN_CAT ('bin_cat_tbl');
   dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
        bin_table_name   => 'bin_cat_tbl',
        data_table_name  => 'mining_data',
        bin_num          =>  3);
  END;
/
DECLARE
  MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name      => 'bin_cat_tbl',
    xform_list          =>  mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list          =>  mining_data_stack,
    data_table_name     => 'mining_data',
    xform_view_name     => 'mining_data_stack_view');
  END;
/
-- Before transformation
column cust_postal_code format a16
SELECT * from mining_data
              WHERE cust_id BETWEEN 100050 AND 100053
              ORDER BY cust_id;

  CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
---------- ---------------- -----------------
    100050 76486                         1500
    100051 73216                         9000
    100052 69499                         5000
    100053 45704                         7000

-- After transformation
SELECT * FROM mining_data_stack_view
              WHERE cust_id BETWEEN 100050 AND 100053
              ORDER BY cust_id;

  CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
---------- ---------------- -----------------
    100050 4                             1500
    100051 1                             9000
    100052 4                             5000
    100053 4                             7000
```

42.2.3.25 STACK_BIN_NUM Procedure

This procedure adds numerical binning transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_NUM (
     bin_table_name     IN             VARCHAR2,
     xform_list         IN OUT  NOCOPY TRANSFORM_LIST,
     literal_flag       IN             BOOLEAN  DEFAULT FALSE,
     bin_schema_name    IN             VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-157 STACK_BIN_NUM Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_BIN_NUM`. To populate the table, you can use one of the `INSERT` procedures for numerical binning or you can write your own SQL. See Table 42-128.
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`literal_flag`	Indicates whether the values in the `bin` column in the transformation definition table are valid SQL literals. When *`literal_flag`* is `FALSE` (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set *`literal_flag`* to `TRUE` if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See "INSERT_BIN_NUM_EQWIDTH Procedure" for an example.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how a binning transformation for the numerical column `cust_credit_limit` could be added to a stack called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_postal_code, cust_credit_limit
      FROM sh.customers
      WHERE cust_id BETWEEN 100050 and 100100;
BEGIN
```

```
  dbms_data_mining_transform.create_bin_num ('bin_num_tbl');
  dbms_data_mining_transform.insert_bin_num_qtile (
  bin_table_name    => 'bin_num_tbl',
  data_table_name   => 'mining_data',
  bin_num               => 5,
  exclude_list      =>  dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/
DECLARE
   MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
   dbms_data_mining_transform.STACK_BIN_CAT (
      bin_table_name        => 'bin_num_tbl',
      xform_list            =>  mining_data_stack);
   dbms_data_mining_transform.XFORM_STACK (
      xform_list            =>  mining_data_stack,
      data_table_name   => 'mining_data',
      xform_view_name   => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit) FROM mining_data
   WHERE cust_id BETWEEN 100050 AND 100055
   ORDER BY cust_id;
CUST_ID   CUST_POSTAL_CODE   ROUND(CUST_CREDIT_LIMIT)
-------   -----------------  -------------------------
100050    76486                                   1500
100051    73216                                   9000
100052    69499                                   5000
100053    45704                                   7000
100055    74673                                  11000
100055    74673                                  11000

-- After transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit)
   FROM mining_data_stack_view
   WHERE cust_id BETWEEN 100050 AND 100055
   ORDER BY cust_id;
CUST_ID   CUST_POSTAL_CODE   ROUND(CUST_CREDIT_LIMITT)
-------   ----------------   -------------------------
100050    76486
100051    73216                                      2
100052    69499                                      1
100053    45704
100054    88021                                      3
100055    74673                                      3
```

42.2.3.26 STACK_CLIP Procedure

This procedure adds clipping transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_CLIP (
          clip_table_name     IN             VARCHAR2,
          xform_list          IN OUT NOCOPY TRANSFORM_LIST,
          clip_schema_name    IN             VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-158 STACK_CLIP Procedure Parameters

Parameter	Description
`clip_table_name`	Name of the transformation definition table for clipping.You can use the CREATE_CLIP Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_CLIP`. To populate the table, you can use one of the `INSERT` procedures for clipping or you can write your own SQL. See Table 42-130
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`clip_schema_name`	Schema of *`clip_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See DBMS_DATA_MINING_TRANSFORM Operational Notes. The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how a clipping transformation for the numerical column `cust_credit_limit` could be added to a stack called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, cust_postal_code, cust_credit_limit
       FROM sh.customers
       WHERE cust_id BETWEEN 100050 AND 100100;
BEGIN
   dbms_data_mining_transform.create_clip ('clip_tbl');
   dbms_data_mining_transform.insert_clip_winsor_tail (
       clip_table_name   => 'clip_tbl',
       data_table_name   => 'mining_data',
       tail_frac         => 0.25,
       exclude_list      => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
```

```
/
DECLARE
      MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
      dbms_data_mining_transform.STACK_CLIP (
         clip_table_name     => 'clip_tbl',
         xform_list          =>  mining_data_stack);
      dbms_data_mining_transform.XFORM_STACK (
          xform_list        =>  mining_data_stack,
          data_table_name   => 'mining_data',
          xform_view_name   => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT cust_id, cust_postal_code, round(cust_credit_limit)
  FROM mining_data
    WHERE cust_id BETWEEN 100050 AND 100054
    ORDER BY cust_id;

CUST_ID   CUST_POSTAL_CODE   ROUND(CUST_CREDIT_LIMIT)
-------   ----------------   ------------------------
100050    76486                                  1500
100051    73216                                  9000
100052    69499                                  5000
100053    45704                                  7000
100054    88021                                 11000

-- After transformation
SELECT cust_id, cust_postal_code, round(cust_credit_limit)
  FROM mining_data_stack_view
    WHERE cust_id BETWEEN 100050 AND 100054
    ORDER BY cust_id;

CUST_ID   CUST_POSTAL_CODE   ROUND(CUST_CREDIT_LIMIT)
-------   ----------------   ------------------------
100050    76486                                  5000
100051    73216                                  9000
100052    69499                                  5000
100053    45704                                  7000
100054    88021                                 11000
```

42.2.3.27 STACK_COL_REM Procedure

This procedure adds column removal transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_COL_REM (
     rem_table_name     IN            VARCHAR2,
     xform_list         IN OUT NOCOPY TRANSFORM_LIST,
     rem_schema_name    IN            VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-159 STACK_COL_REM Procedure Parameters

Parameter	Description
`rem_table_name`	Name of the transformation definition table for column removal. You can use the CREATE_COL_REM Procedure to create the definition table. See Table 42-132. The table must be populated with column names before you call `STACK_COL_REM`. The INSERT_BIN_SUPER Procedure and the INSERT_AUTOBIN_NUM_EQWIDTH Procedure can optionally be used to populate the table. You can also use SQL `INSERT` statements.
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`rem_schema_name`	Schema of *`rem_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the column `cust_credit_limit` could be removed in a transformation list called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
     SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
        FROM sh.customers;

BEGIN
    dbms_data_mining_transform.create_col_rem ('rem_tbl');
END;
/

INSERT into rem_tbl VALUES (upper('cust_postal_code'), null);

DECLARE
```

```
  MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.stack_col_rem (
          rem_table_name     => 'rem_tbl',
          xform_list         =>  mining_data_stack);
      dbms_data_mining_transform.XFORM_STACK (
          xform_list         =>  mining_data_stack,
          data_table_name    => 'mining_data',
          xform_view_name    => 'mining_data_stack_view');
END;
/

SELECT *  FROM mining_data
  WHERE cust_id BETWEEN 100050 AND 100051
  ORDER BY cust_id;

CUST_ID   COUNTRY_ID   CUST_POSTAL_CODE   CUST_CREDIT_LIMIT
-------   ----------   ----------------   -----------------
100050         52773   76486                           1500
100051         52790   73216                           9000

SELECT *  FROM mining_data_stack_view
  WHERE cust_id BETWEEN 100050 AND 100051
  ORDER BY cust_id;

CUST_ID   COUNTRY_ID   CUST_CREDIT_LIMIT
-------   ----------   -----------------
100050         52773                1500
100051         52790                9000
```

42.2.3.28 STACK_MISS_CAT Procedure

This procedure adds categorical missing value transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_MISS_CAT (
     miss_table_name     IN       VARCHAR2,
     xform_list          IN OUT   NOCOPY TRANSFORM_LIST,
     miss_schema_name    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-160 STACK_MISS_CAT Procedure Parameters

Parameter	Description
`miss_table_name`	Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_MISS_CAT`. To populate the table, you can use the INSERT_MISS_CAT_MODE Procedure or you can write your own SQL. See Table 42-134.
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`miss_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the missing values in the column `cust_marital_status` could be replaced with the mode in a transformation list called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
      SELECT cust_id, country_id, cust_marital_status
         FROM sh.customers
         where cust_id BETWEEN 1 AND 10;

BEGIN
  dbms_data_mining_transform.create_miss_cat ('miss_cat_tbl');
  dbms_data_mining_transform.insert_miss_cat_mode ('miss_cat_tbl', 'mining_data');
END;
/

DECLARE
  MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
     dbms_data_mining_transform.stack_miss_cat (
          miss_table_name   => 'miss_cat_tbl',
          xform_list        => mining_data_stack);
      dbms_data_mining_transform.XFORM_STACK (
          xform_list        =>  mining_data_stack,
          data_table_name   => 'mining_data',
          xform_view_name   => 'mining_data_stack_view');
END;
/
SELECT * FROM mining_data
  ORDER BY cust_id;

CUST_ID   COUNTRY_ID   CUST_MARITAL_STATUS
-------   ----------   --------------------
      1      52789
      2      52778
      3      52770
      4      52770
```

```
      5      52789
      6      52769      single
      7      52790      single
      8      52790      married
      9      52770      divorced
     10      52790      widow

SELECT * FROM mining_data_stack_view
   ORDER By cust_id;

CUST_ID   COUNTRY_ID   CUST_MARITAL_STATUS
-------   -----------  --------------------
      1       52789    single
      2       52778    single
      3       52770    single
      4       52770    single
      5       52789    single
      6       52769    single
      7       52790    single
      8       52790    married
      9       52770    divorced
     10       52790    widow
```

42.2.3.29 STACK_MISS_NUM Procedure

This procedure adds numeric missing value transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_MISS_NUM (
     miss_table_name     IN       VARCHAR2,
     xform_list          IN OUT   NOCOPY TRANSFORM_LIST,
     miss_schema_name    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-161 STACK_MISS_NUM Procedure Parameters

Parameter	Description
`miss_table_name`	Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_MISS_NUM`. To populate the table, you can use the INSERT_MISS_NUM_MEAN Procedure or you can write your own SQL. See Table 42-136.
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`miss_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"

- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the missing values in the column `cust_credit_limit` could be replaced with the mean in a transformation list called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
describe mining_data
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- -----
 CUST_ID                                               NOT NULL NUMBER
 CUST_CREDIT_LIMIT                                              NUMBER

BEGIN
   dbms_data_mining_transform.create_miss_num ('miss_num_tbl');
   dbms_data_mining_transform.insert_miss_num_mean ('miss_num_tbl','mining_data');
END;
/
SELECT * FROM miss_num_tbl;

COL                  ATT      VAL
-------------------- ----- ------
CUST_ID                       5.5
CUST_CREDIT_LIMIT          185.71

DECLARE
    MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.STACK_MISS_NUM (
          miss_table_name   => 'miss_num_tbl',
          xform_list        => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
          xform_list        =>  mining_data_stack,
          data_table_name   => 'mining_data',
          xform_view_name   => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT * FROM mining_data
  ORDER BY cust_id;
CUST_ID CUST_CREDIT_LIMIT
------- -----------------
      1               100
      2
      3               200
      4
```

```
      5              150
      6              400
      7              150
      8
      9              100
     10              200

-- After transformation
SELECT * FROM mining_data_stack_view
  ORDER BY cust_id;
CUST_ID CUST_CREDIT_LIMIT
------- -----------------
      1              100
      2           185.71
      3              200
      4           185.71
      5              150
      6              400
      7              150
      8           185.71
      9              100
     10              200
```

42.2.3.30 STACK_NORM_LIN Procedure

This procedure adds linear normalization transformations to a transformation list.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_NORM_LIN (
     norm_table_name     IN       VARCHAR2,
     xform_list          IN OUT   NOCOPY TRANSFORM_LIST,
     norm_schema_name    IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-162 STACK_NORM_LIN Procedure Parameters

Parameter	Description
`norm_table_name`	Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The table must be populated with transformation definitions before you call `STACK_NORM_LIN`.To populate the table, you can use one of the `INSERT` procedures for normalization or you can write your own SQL. See Table 42-138.
`xform_list`	A transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`norm_schema_name`	Schema of *`norm_table_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"

- "About Stacking"
- "Nested Data Transformations"

Examples

This example shows how the column `cust_credit_limit` could be normalized in a transformation list called `mining_data_stack`.

> **Note:**
>
> This example invokes the XFORM_STACK Procedure to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See INSERT_BIN_NUM_EQWIDTH Procedure for an example.

```
CREATE OR REPLACE VIEW mining_data AS
       SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
          FROM sh.customers;
BEGIN
   dbms_data_mining_transform.create_norm_lin ('norm_lin_tbl');
   dbms_data_mining_transform.insert_norm_lin_minmax (
       norm_table_name   => 'norm_lin_tbl',
       data_table_name   => 'mining_data',
       exclude_list      =>  dbms_data_mining_transform.COLUMN_LIST('cust_id',
                                                                'country_id'));
END;
/
SELECT * FROM norm_lin_tbl;
COL                    ATT    SHIFT  SCALE
-------------------- ----- ------ ------
CUST_CREDIT_LIMIT             1500  13500

DECLARE
   MINING_DATA_STACK   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
   dbms_data_mining_transform.stack_norm_lin (
        norm_table_name   => 'norm_lin_tbl',
        xform_list        => mining_data_stack);
   dbms_data_mining_transform.XFORM_STACK (
        xform_list        =>  mining_data_stack,
        data_table_name   => 'mining_data',
        xform_view_name   => 'mining_data_stack_view');
END;
/
SELECT * FROM mining_data
  WHERE cust_id between 1 and 10
  ORDER BY cust_id;
CUST_ID COUNTRY_ID CUST_POSTAL_CODE     CUST_CREDIT_LIMIT
------- ---------- -------------------- -----------------
      1      52789 30828                             9000
      2      52778 86319                            10000
      3      52770 88666                             1500
      4      52770 87551                             1500
      5      52789 59200                             1500
```

```
      6      52769 77287                           1500
      7      52790 38763                           1500
      8      52790 58488                           3000
      9      52770 63033                           3000
     10      52790 52602                           3000

SELECT * FROM mining_data_stack_view
  WHERE cust_id between 1 and 10
  ORDER BY cust_id;
CUST_ID COUNTRY_ID CUST_POSTAL_CODE     CUST_CREDIT_LIMIT
------- ---------- -------------------- -----------------
      1      52789 30828                           .55556
      2      52778 86319                           .62963
      3      52770 88666                                0
      4      52770 87551                                0
      5      52789 59200                                0
      6      52769 77287                                0
      7      52790 38763                                0
      8      52790 58488                           .11111
      9      52770 63033                           .11111
     10      52790 52602                           .11111
```

42.2.3.31 XFORM_BIN_CAT Procedure

This procedure creates a view that implements the categorical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_CAT (
     bin_table_name        IN VARCHAR2,
     data_table_name       IN VARCHAR2,
     xform_view_name       IN VARCHAR2,
     literal_flag          IN BOOLEAN DEFAULT FALSE,
     bin_schema_name       IN VARCHAR2 DEFAULT NULL,
     data_schema_name      IN VARCHAR2 DEFAULT NULL,
     xform_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-163 XFORM_BIN_CAT Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table for categorical binning. You can use the CREATE_BIN_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call `XFORM_BIN_CAT`. To populate the table, you can use one of the `INSERT` procedures for categorical binning or you can write your own SQL. See Table 42-126.
`data_table_name`	Name of the table containing the data to be transformed.
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`bin_table_name`*.

Table 42-163 (Cont.) XFORM_BIN_CAT Procedure Parameters

Parameter	Description
`literal_flag`	Indicates whether the values in the `bin` column in the transformation definition table are valid SQL literals. When *`literal_flag`* is `FALSE` (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set *`literal_flag`* to `TRUE` if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See "INSERT_BIN_NUM_EQWIDTH Procedure" for an example.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that bins the `cust_postal_code` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_POSTAL_CODE                          NOT NULL VARCHAR2(10)
 CUST_CREDIT_LIMIT                                  NUMBER

SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;

   CUST_ID CUST_POSTAL_CODE
CUST_CREDIT_LIMIT
--------- --------------------
-----------------
    104066 69776
7000
    104067 52602
9000
    104068 55787
11000
    104069 55977
5000

BEGIN
  dbms_data_mining_transform.create_bin_cat(
     bin_table_name     => 'bin_cat_tbl');
  dbms_data_mining_transform.insert_bin_cat_freq(
     bin_table_name     => 'bin_cat_tbl',
     data_table_name    => 'mining_data',
     bin_num          => 10);
```

```
   dbms_data_mining_transform.xform_bin_cat(
     bin_table_name     => 'bin_cat_tbl',
     data_table_name    => 'mining_data',
     xform_view_name    => 'bin_cat_view');
END;
/

SELECT * FROM bin_cat_view WHERE cust_id between 104066 and 104069;

   CUST_ID CUST_POSTAL_CODE
CUST_CREDIT_LIMIT
---------- --------------------
-----------------
    104066 6
7000
    104067 11
9000
    104068 3
11000
    104069 11
5000

SELECT text FROM user_views WHERE view_name IN 'BIN_CAT_VIEW';

TEXT

--------------------------------------------------------------------------------

SELECT
"CUST_ID",DECODE("CUST_POSTAL_CODE",'38082','1','45704','9','48346','5','

55787','3','63736','2','67843','7','69776','6','72860','10','78558','4','80841',

'8',NULL,NULL,'11') "CUST_POSTAL_CODE","CUST_CREDIT_LIMIT" FROM
mining_data
```

42.2.3.32 XFORM_BIN_NUM Procedure

This procedure creates a view that implements the numerical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM (
     bin_table_name      IN VARCHAR2,
     data_table_name     IN VARCHAR2,
     xform_view_name     IN VARCHAR2,
     literal_flag        IN BOOLEAN DEFAULT FALSE,
     bin_schema_name     IN VARCHAR2 DEFAULT NULL,
     data_schema_name    IN VARCHAR2 DEFAULT NULL,
     xform_schema_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-164 XFORM_BIN_NUM Procedure Parameters

Parameter	Description
`bin_table_name`	Name of the transformation definition table for numerical binning. You can use the CREATE_BIN_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call `XFORM_BIN_NUM`. To populate the table, you can use one of the `INSERT` procedures for numerical binning or you can write your own SQL. See "Table 42-128".
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`bin_table_name`*.
`literal_flag`	Indicates whether the values in the `bin` column in the transformation definition table are valid SQL literals. When *`literal_flag`* is `FALSE` (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes. Set *`literal_flag`* to `TRUE` if the bin identifiers are numbers that should have a numeric datatype, as is the case for an O-Cluster model. See "INSERT_BIN_NUM_EQWIDTH Procedure" for an example.
`bin_schema_name`	Schema of *`bin_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that bins the `cust_credit_limit` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_POSTAL_CODE                          NOT NULL VARCHAR2(10)
 CUST_CREDIT_LIMIT                                  NUMBER

column cust_credit_limit off
SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;

   CUST_ID CUST_POSTAL_CODE
CUST_CREDIT_LIMIT
---------  ------------------
--------------------
    104066 69776
7000
    104067 52602
```

```
9000
    104068 55787
11000
    104069 55977
5000

BEGIN
   dbms_data_mining_transform.create_bin_num(
           bin_table_name     => 'bin_num_tbl');
   dbms_data_mining_transform.insert_autobin_num_eqwidth(
           bin_table_name     => 'bin_num_tbl',
           data_table_name    => 'mining_data',
           bin_num                => 5,
           max_bin_num            => 10,
           exclude_list       =>
dbms_data_mining_transform.COLUMN_LIST('cust_id'));
  dbms_data_mining_transform.xform_bin_num(
          bin_table_name      => 'bin_num_tbl',
          data_table_name     => 'mining_data',
          xform_view_name     => 'mining_data_view');
END;
/
describe mining_data_view
 Name                                 Null?    Type
 ------------------------------------ -------- ------------------------
 CUST_ID                              NOT NULL NUMBER
 CUST_POSTAL_CODE                     NOT NULL VARCHAR2(10)
 CUST_CREDIT_LIMIT                             VARCHAR2(2)

col cust_credit_limit on
col cust_credit_limit format a25
SELECT * FROM mining_data_view WHERE cust_id between 104066 and 104069;

   CUST_ID CUST_POSTAL_CODE
CUST_CREDIT_LIMIT
---------- --------------------
-------------------------
    104066 69776
5
    104067 52602
6
    104068 55787
8
    104069 55977
3

set long 2000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_VIEW';

TEXT

--------------------------------------------------------------------------------

SELECT "CUST_ID","CUST_POSTAL_CODE",CASE WHEN "CUST_CREDIT_LIMIT"<1500 THEN
NULL
 WHEN "CUST_CREDIT_LIMIT"<=2850 THEN '1' WHEN "CUST_CREDIT_LIMIT"<=4200 THEN
'2'
 WHEN "CUST_CREDIT_LIMIT"<=5550 THEN '3' WHEN "CUST_CREDIT_LIMIT"<=6900 THEN
'4'
 WHEN "CUST_CREDIT_LIMIT"<=8250 THEN '5' WHEN "CUST_CREDIT_LIMIT"<=9600 THEN
'6'
```

```
 WHEN "CUST_CREDIT_LIMIT"<=10950 THEN '7' WHEN "CUST_CREDIT_LIMIT"<=12300 THEN
'
8' WHEN "CUST_CREDIT_LIMIT"<=13650 THEN '9' WHEN "CUST_CREDIT_LIMIT"<=15000
THEN
 '10' END "CUST_CREDIT_LIMIT" FROM
mining_data
```

42.2.3.33 XFORM_CLIP Procedure

This procedure creates a view that implements the clipping transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_CLIP (
    clip_table_name        IN VARCHAR2,
    data_table_name        IN VARCHAR2,
    xform_view_name        IN VARCHAR2,
    clip_schema_name       IN VARCHAR2 DEFAULT NULL,
    data_schema_name       IN VARCHAR2,DEFAULT NULL,
    xform_schema_name      IN VARCHAR2,DEFAULT NULL);
```

Parameters

Table 42-165 XFORM_CLIP Procedure Parameters

Parameter	Description
`clip_table_name`	Name of the transformation definition table for clipping. You can use the CREATE_CLIP Procedure to create the definition table. The table must be populated with transformation definitions before you call `XFORM_CLIP`. To populate the table, you can use one of the `INSERT` procedures for clipping you can write your own SQL. See Table 42-130.
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`clip_table_name`*.
`clip_schema_name`	Schema of *`clip_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Examples

This example creates a view that clips the `cust_credit_limit` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
 Name                            Null?    Type
 ------------------------------- -------- -------------------------
 CUST_ID                         NOT NULL NUMBER
 CUST_POSTAL_CODE                NOT NULL VARCHAR2(10)
```

```
 CUST_CREDIT_LIMIT                        NUMBER

BEGIN
   dbms_data_mining_transform.create_clip(
      clip_table_name    => 'clip_tbl');
   dbms_data_mining_transform.insert_clip_trim_tail(
      clip_table_name   => 'clip_tbl',
      data_table_name   => 'mining_data',
      tail_frac         => 0.05,
      exclude_list      => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
   dbms_data_mining_transform.xform_clip(
      clip_table_name     => 'clip_tbl',
      data_table_name    => 'mining_data',
      xform_view_name    => 'clip_view');
END;
/
describe clip_view
 Name                             Null?    Type
 ----------------------------- -------- --------------------------
 CUST_ID                          NOT NULL NUMBER
 CUST_POSTAL_CODE                 NOT NULL VARCHAR2(10)
 CUST_CREDIT_LIMIT                         NUMBER

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM mining_data;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
---------------------- ----------------------
                  1500                  15000

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM clip_view;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
---------------------- ----------------------
                  1500                  11000

set long 2000
SELECT text FROM user_views WHERE view_name IN 'CLIP_VIEW';

TEXT
--------------------------------------------------------------------------------
SELECT "CUST_ID","CUST_POSTAL_CODE",CASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NU
LL WHEN "CUST_CREDIT_LIMIT" > 11000 THEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST
_CREDIT_LIMIT" FROM mining_data
```

42.2.3.34 XFORM_COL_REM Procedure

This procedure creates a view that implements the column removal transformations specified in a definition table. Only the columns that are specified in the definition table are removed; the remaining columns from the data table are present in the view.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (
     rem_table_name      IN       VARCHAR2,
     data_table_name     IN       VARCHAR2,
     xform_view_name     IN       VARCHAR2,
     rem_schema_name     IN       VARCHAR2 DEFAULT NULL,
     data_schema_name    IN       VARCHAR2 DEFAULT NULL,
     xform_schema_name   IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-166 XFORM_COL_REM Procedure Parameters

Parameter	Description
`rem_table_name`	Name of the transformation definition table for column removal. You can use the CREATE_COL_REM Procedure to create the definition table. See Table 42-132. The table must be populated with column names before you call `XFORM_COL_REM`. The INSERT_BIN_SUPER Procedure and the INSERT_AUTOBIN_NUM_EQWIDTH Procedure can optionally be used to populate the table. You can also use SQL `INSERT` statements.
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents the columns in *`data_table_name`* that are not specified in *`rem_table_name`*.
`rem_schema_name`	Schema of *`rem_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that includes all but one column from the table `customers` in the current schema.

```
describe customers
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_MARITAL_STATUS                                VARCHAR2(20)
 OCCUPATION                                         VARCHAR2(21)
 AGE                                                NUMBER
 YRS_RESIDENCE                                      NUMBER

BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('colrem_xtbl');
END;
 /
INSERT INTO colrem_xtbl VALUES('CUST_MARITAL_STATUS', null);

NOTE: This currently doesn't work. See bug 9310319

BEGIN
   DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (
     rem_table_name        => 'colrem_xtbl',
     data_table_name       => 'customers',
     xform_view_name       => 'colrem_view');
END;
/
```

```
describe colrem_view

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 OCCUPATION                                         VARCHAR2(21)
 AGE                                                NUMBER
 YRS_RESIDENCE                                      NUMBER
```

42.2.3.35 XFORM_EXPR_NUM Procedure

This procedure creates a view that implements the specified numeric transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_NUM (
     expr_pattern        IN       VARCHAR2,
     data_table_name     IN       VARCHAR2,
     xform_view_name     IN       VARCHAR2,
     exclude_list        IN       COLUMN_LIST DEFAULT NULL,
     include_list        IN       COLUMN_LIST DEFAULT NULL,
     col_pattern         IN       VARCHAR2 DEFAULT ':col',
     data_schema_name    IN       VARCHAR2 DEFAULT NULL,
     xform_schema_name   IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-167 XFORM_EXPR_NUM Procedure Parameters

Parameter	Description
`expr_pattern`	A numeric transformation expression
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`expr_pattern`* and *`col_pattern`*.
`exclude_list`	List of numerical columns to exclude. If `NULL`, no numerical columns are excluded. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`include_list`	List of numeric columns to include. If `NULL`, all numeric columns are included. The format of *`include_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`col_pattern`	The value within *`expr_pattern`* that will be replaced with a column name. The value of *`col_pattern`* is case-sensitive. The default value of *`col_pattern`* is `':col'`

Table 42-167 (Cont.) XFORM_EXPR_NUM Procedure Parameters

Parameter	Description
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.
xform_schema_name	Schema of *xform_view_name*. If no schema is specified, the current schema is used.

Usage Notes

1. The XFORM_EXPR_NUM procedure constructs numeric transformation expressions from the specified expression pattern (*expr_pattern*) by replacing every occurrence of the specified column pattern (*col_pattern*) with an actual column name.

   XFORM_EXPR_NUM uses the SQL REPLACE function to construct the transformation expressions.

   ```
   REPLACE (expr_pattern,col_pattern,'"column_name"') || '"column_name"'
   ```

   If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

   > **See:**
   >
   > *Oracle Database SQL Language Reference* for information about the REPLACE function

2. Because of the include and exclude list parameters, the XFORM_EXPR_NUM and XFORM_EXPR_STR procedures allow you to easily specify individual columns for transformation within large data sets. The other XFORM_* procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.
3. See "Operational Notes"

Examples

This example creates a view that transforms the datatype of numeric columns.

```
describe customers
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_MARITAL_STATUS                                VARCHAR2(20)
 OCCUPATION                                         VARCHAR2(21)
 AGE                                                NUMBER
 YRS_RESIDENCE                                      NUMBER

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_NUM(
    expr_pattern          => 'to_char(:col)',
    data_table_name       => 'customers',
    xform_view_name       => 'cust_nonum_view',
    exclude_list          => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
```

```
    include_list         => null,
    col_pattern          => ':col');
END;
/
describe cust_nonum_view
 Name                                   Null?    Type
 ----------------------------------- -------- ------------------------
 CUST_ID                                NOT NULL NUMBER
 CUST_MARITAL_STATUS                             VARCHAR2(20)
 OCCUPATION                                      VARCHAR2(21)
 AGE                                             VARCHAR2(40)
 YRS_RESIDENCE                                   VARCHAR2(40)
```

42.2.3.36 XFORM_EXPR_STR Procedure

This procedure creates a view that implements the specified categorical transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR (
     expr_pattern       IN       VARCHAR2,
     data_table_name    IN       VARCHAR2,
     xform_view_name    IN       VARCHAR2,
     exclude_list       IN       COLUMN_LIST DEFAULT NULL,
     include_list       IN       COLUMN_LIST DEFAULT NULL,
     col_pattern        IN       VARCHAR2 DEFAULT ':col',
     data_schema_name   IN       VARCHAR2 DEFAULT NULL,
     xform_schema_name  IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-168 XFORM_EXPR_STR Procedure Parameters

Parameter	Description
`expr_pattern`	A character transformation expression
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`expr_pattern`* and *`col_pattern`*.
`exclude_list`	List of categorical columns to exclude. If `NULL`, no categorical columns are excluded. The format of *`exclude_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`
`include_list`	List of character columns to include. If `NULL`, all character columns are included. The format of *`include_list`* is: `dbms_data_mining_transform.COLUMN_LIST('col1','col2', ...'coln')`

Table 42-168 (Cont.) XFORM_EXPR_STR Procedure Parameters

Parameter	Description
`col_pattern`	The value within *`expr_pattern`* that will be replaced with a column name. The value of *`col_pattern`* is case-sensitive. The default value of *`col_pattern`* is `':col'`
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

1. The `XFORM_EXPR_STR` procedure constructs character transformation expressions from the specified expression pattern (*`expr_pattern`*) by replacing every occurrence of the specified column pattern (*`col_pattern`*) with an actual column name.

   `XFORM_EXPR_STR` uses the SQL `REPLACE` function to construct the transformation expressions.

   ```
   REPLACE (expr_pattern,col_pattern,'"column_name"') || '"column_name"'
   ```

   If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

   > **See:**
   >
   > *Oracle Database SQL Language Reference* for information about the `REPLACE` function

2. Because of the include and exclude list parameters, the `XFORM_EXPR_STR` and `XFORM_EXPR_NUM` procedures allow you to easily specify individual columns for transformation within large data sets. The other `XFORM_*` procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.
3. See "Operational Notes"

Examples

This example creates a view that transforms character columns to upper case.

```
describe customers
 Name                                    Null?    Type
 ----------------------------------- -------- ------------------------
 CUST_ID                                 NOT NULL NUMBER
 CUST_MARITAL_STATUS                              VARCHAR2(20)
 OCCUPATION                                       VARCHAR2(21)
 AGE                                              NUMBER
 YRS_RESIDENCE                                    NUMBER
```

```
SELECT cust_id,  cust_marital_status, occupation FROM customers
    WHERE   cust_id > 102995
    ORDER BY cust_id desc;

CUST_ID CUST_MARITAL_STATUS  OCCUPATION
------- -------------------- ---------------------
 103000 Divorc.              Cleric.
 102999 Married              Cleric.
 102998 Married              Exec.
 102997 Married              Exec.
 102996 NeverM               Other

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR(
       expr_pattern            => 'upper(:col)',
       data_table_name         => 'customers',
       xform_view_name         => 'cust_upcase_view');
END;
/
describe cust_upcase_view
 Name                          Null?    Type
 ----------------------------- -------- --------------------
 CUST_ID                       NOT NULL NUMBER
 CUST_MARITAL_STATUS                    VARCHAR2(20)
 OCCUPATION                             VARCHAR2(21)
 AGE                                    NUMBER
 YRS_RESIDENCE                          NUMBER

SELECT cust_id,  cust_marital_status, occupation FROM cust_upcase_view
   WHERE   cust_id > 102995
   ORDER BY cust_id desc;

CUST_ID CUST_MARITAL_STATUS  OCCUPATION
------- -------------------- ---------------------
 103000 DIVORC.              CLERIC.
 102999 MARRIED              CLERIC.
 102998 MARRIED              EXEC.
 102997 MARRIED              EXEC.
 102996 NEVERM               OTHER
```

42.2.3.37 XFORM_MISS_CAT Procedure

This procedure creates a view that implements the categorical missing value treatment transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (
    miss_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    xform_schema_name     IN VARCHAR2 DEFAULT NULL;
```

Parameters

Table 42-169 XFORM_MISS_CAT Procedure Parameters

Parameter	Description
`miss_table_name`	Name of the transformation definition table for categorical missing value treatment. You can use the CREATE_MISS_CAT Procedure to create the definition table. The table must be populated with transformation definitions before you call `XFORM_MISS_CAT`. To populate the table, you can use the INSERT_MISS_CAT_MODE Procedure or you can write your own SQL. See Table 42-134.
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`miss_table_name`*.
`miss_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that replaces missing categorical values with the mode.

```
SELECT * FROM geog;

REG_ID REGION
------ ------------------------------
     1 NE
     2 SW
     3 SE
     4 SW
     5
     6 NE
     7 NW
     8 NW
     9
    10
    11 SE
    12 SE
    13 NW
    14 SE
    15 SE

SELECT STATS_MODE(region) FROM geog;

STATS_MODE(REGION)
------------------------------
SE
```

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('misscat_xtbl');
  DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (
    miss_table_name         => 'misscat_xtbl',
    data_table_name         => 'geog' );
END;
/

SELECT col, val FROM misscat_xtbl;

COL        VAL
---------- ----------
REGION     SE

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (
     miss_table_name        => 'misscat_xtbl',
     data_table_name        => 'geog',
     xform_view_name        => 'geogxf_view');
END;
/

SELECT * FROM geogxf_view;

REG_ID REGION
------ ------------------------------
     1 NE
     2 SW
     3 SE
     4 SW
     5 SE
     6 NE
     7 NW
     8 NW
     9 SE
    10 SE
    11 SE
    12 SE
    13 NW
    14 SE
    15 SE
```

42.2.3.38 XFORM_MISS_NUM Procedure

This procedure creates a view that implements the numerical missing value treatment transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
    miss_table_name       IN VARCHAR2,
    data_table_name       IN VARCHAR2,
    xform_view_name       IN VARCHAR2,
    miss_schema_name      IN VARCHAR2 DEFAULT NULL,
    data_schema_name      IN VARCHAR2 DEFAULT NULL,
    xform_schema_name     IN VARCHAR2 DEFAULT NULL;
```

Parameters

Table 42-170 XFORM_MISS_NUM Procedure Parameters

Parameter	Description
miss_table_name	Name of the transformation definition table for numerical missing value treatment. You can use the CREATE_MISS_NUM Procedure to create the definition table. The table must be populated with transformation definitions before you call XFORM_MISS_NUM. To populate the table, you can use the INSERT_MISS_NUM_MEAN Procedure or you can write your own SQL. See Table 42-136.
data_table_name	Name of the table containing the data to be transformed
xform_view_name	Name of the view to be created. The view presents columns in *data_table_name* with the transformations specified in *miss_table_name*.
miss_schema_name	Schema of *miss_table_name*. If no schema is specified, the current schema is used.
data_schema_name	Schema of *data_table_name*. If no schema is specified, the current schema is used.
xform_schema_name	Schema of *xform_view_name*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that replaces missing numerical values with the mean.

```
SELECT * FROM items;

ITEM_ID       QTY
---------- ------
aa            200
bb            200
cc            250
dd
ee
ff            100
gg            250
hh            200
ii
jj            200

SELECT AVG(qty) FROM items;

AVG(QTY)
--------
     200

BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('missnum_xtbl');
  DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
```

```
     miss_table_name          => 'missnum_xtbl',
     data_table_name         => 'items' );
END;
/

SELECT col, val FROM missnum_xtbl;

COL           VAL
---------- ------
QTY           200

BEGIN
    DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
        miss_table_name        => 'missnum_xtbl',
        data_table_name        => 'items',
        xform_view_name        => 'items_view');
END;
/

SELECT * FROM items_view;

ITEM_ID       QTY
---------- ------
aa            200
bb            200
cc            250
dd            200
ee            200
ff            100
gg            250
hh            200
ii            200
jj            200
```

42.2.3.39 XFORM_NORM_LIN Procedure

This procedure creates a view that implements the linear normalization transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
     norm_table_name      IN VARCHAR2,
     data_table_name      IN VARCHAR2,
     xform_view_name      IN VARCHAR2,
     norm_schema_name     IN VARCHAR2 DEFAULT NULL,
     data_schema_name     IN VARCHAR2 DEFAULT NULL,
     xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-171 XFORM_NORM_LIN Procedure Parameters

Parameter	Description
`norm_table_name`	Name of the transformation definition table for linear normalization. You can use the CREATE_NORM_LIN Procedure to create the definition table. The table must be populated with transformation definitions before you call `XFORM_NORM_LIN`. To populate the table, you can use one of the `INSERT` procedures for normalization or you can write your own SQL. See Table 42-134.
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view presents columns in *`data_table_name`* with the transformations specified in *`miss_table_name`*.
`norm_schema_name`	Schema of *`miss_table_name`*. If no schema is specified, the current schema is used.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes".

Examples

This example creates a view that normalizes the `cust_year_of_birth` and `cust_credit_limit` columns. The data source consists of three columns from `sh.customer`.

```
CREATE OR REPLACE VIEW mining_data AS
     SELECT cust_id, cust_year_of_birth, cust_credit_limit
     FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_CREDIT_LIMIT                                  NUMBER

SELECT * FROM mining_data WHERE cust_id > 104495
      ORDER BY cust_year_of_birth;

 CUST_ID CUST_YEAR_OF_BIRTH CUST_CREDIT_LIMIT
-------- ------------------ -----------------
  104496               1947              3000
  104498               1954             10000
  104500               1962             15000
  104499               1970              3000
  104497               1976              3000

BEGIN
  dbms_data_mining_transform.CREATE_NORM_LIN(
```

```
       norm_table_name        => 'normx_tbl');
 dbms_data_mining_transform.INSERT_NORM_LIN_MINMAX(
      norm_table_name     => 'normx_tbl',
      data_table_name     => 'mining_data',
      exclude_list     => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
      round_num      => 3);
END;
/

SELECT col, shift, scale FROM normx_tbl;

COL                                  SHIFT    SCALE
------------------------------ -------- --------
CUST_YEAR_OF_BIRTH                    1910       77
CUST_CREDIT_LIMIT                     1500    13500

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
     norm_table_name     => 'normx_tbl',
     data_table_name     => 'mining_data',
     xform_view_name     => 'norm_view');
END;
/

SELECT * FROM norm_view WHERE cust_id > 104495
      ORDER BY cust_year_of_birth;

 CUST_ID CUST_YEAR_OF_BIRTH CUST_CREDIT_LIMIT
-------- ------------------ -----------------
  104496           .4805195          .1111111
  104498           .5714286          .6296296
  104500           .6753247                 1
  104499           .7792208          .1111111
  104497           .8571429          .1111111


set long 2000
SQL> SELECT text FROM user_views WHERE view_name IN 'NORM_VIEW';

TEXT
---------------------------------------------------------------------------
SELECT "CUST_ID",("CUST_YEAR_OF_BIRTH"-1910)/77 "CUST_YEAR_OF_BIRTH",("CUST
_CREDIT_LIMIT"-1500)/13500 "CUST_CREDIT_LIMIT" FROM mining_data
```

42.2.3.40 XFORM_STACK Procedure

This procedure creates a view that implements the transformations specified by the stack. Only the columns and nested attributes that are specified in the stack are transformed. Any remaining columns and nested attributes from the data table appear in the view without changes.

To create a list of objects that describe the transformed columns, use the DESCRIBE_STACK Procedure.

See Also:

"Overview"

Oracle Machine Learning for SQL User's Guide for more information about machine learning attributes

Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_STACK (
     xform_list          IN     TRANSFORM_list,
     data_table_name     IN     VARCHAR2,
     xform_view_name     IN     VARCHAR2,
     data_schema_name    IN     VARCHAR2 DEFAULT NULL,
     xform_schema_name   IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-172 XFORM_STACK Procedure Parameters

Parameter	Description
`xform_list`	The transformation list. See Table 42-123 for a description of the `TRANSFORM_LIST` object type.
`data_table_name`	Name of the table containing the data to be transformed
`xform_view_name`	Name of the view to be created. The view applies the transformations in *`xform_list`* to *`data_table_name`*.
`data_schema_name`	Schema of *`data_table_name`*. If no schema is specified, the current schema is used.
`xform_schema_name`	Schema of *`xform_view_name`*. If no schema is specified, the current schema is used.

Usage Notes

See "Operational Notes". The following sections are especially relevant:

- "About Transformation Lists"
- "About Stacking"
- "Nested Data Transformations"

Examples

This example applies a transformation list to the view `oml_user.cust_info` and shows how the data is transformed. The `CREATE` statement for `cust_info` is shown in "DESCRIBE_STACK Procedure".

```
BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM ('birth_yr_bins');
  dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (
        bin_table_name    => 'birth_yr_bins',
        data_table_name   =>  'cust_info',
        bin_num           =>   6,
        exclude_list      =>  dbms_data_mining_transform.column_list(
```

```
                                  'cust_id','country_id'));
END;
/
SELECT * FROM birth_yr_bins;

COL                   ATT      VAL BIN
-------------------- ----- ------ ----------
CUST_YEAR_OF_BIRTH           1922
CUST_YEAR_OF_BIRTH           1951 1
CUST_YEAR_OF_BIRTH           1959 2
CUST_YEAR_OF_BIRTH           1966 3
CUST_YEAR_OF_BIRTH           1973 4
CUST_YEAR_OF_BIRTH           1979 5
CUST_YEAR_OF_BIRTH           1986 6

DECLARE
      cust_stack   dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
      dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
          'country_id', NULL, 'country_id/10', 'country_id*10');
      dbms_data_mining_transform.STACK_BIN_NUM ('birth_yr_bins',
          cust_stack);
      dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
          'custprods', 'Mouse Pad', 'value*100', 'value/100');
      dbms_data_mining_transform.XFORM_STACK(
          xform_list        => cust_stack,
          data_table_name   => 'cust_info',
          xform_view_name   =>  'cust_xform_view');
  END;
/

-- Two rows of data without transformations
SELECT * from cust_info WHERE cust_id BETWEEN 100010 AND 100011;

CUST_ID COUNTRY_ID CUST_YEAR_OF_BIRTH CUSTPRODS(ATTRIBUTE_NAME, VALUE)
------- ---------- ------------------ -----------------------------------------
 100010      52790               1975 DM_NESTED_NUMERICALS(
                                        DM_NESTED_NUMERICAL(
                                         '18" Flat Panel Graphics Monitor', 1),
                                        DM_NESTED_NUMERICAL(
                                         'SIMM- 16MB PCMCIAII card', 1))
 100011      52775               1972 DM_NESTED_NUMERICALS(
                                       DM_NESTED_NUMERICAL(
                                         'External 8X CD-ROM', 1),
                                       DM_NESTED_NUMERICAL(
                                         'Mouse Pad', 1),
                                       DM_NESTED_NUMERICAL(
                                        'SIMM- 16MB PCMCIAII card', 1),
                                       DM_NESTED_NUMERICAL(
                                         'Keyboard Wrist Rest', 1),
                                       DM_NESTED_NUMERICAL(
                                         '18" Flat Panel Graphics Monitor', 1),
                                       DM_NESTED_NUMERICAL(
                                         'O/S Documentation Set - English', 1))

-- Same two rows of data with transformations
SELECT * FROM cust_xform_view WHERE cust_id BETWEEN 100010 AND 100011;

CUST_ID   COUNTRY_ID  C  CUSTPRODS(ATTRIBUTE_NAME, VALUE)
-------   ----------  -  --------------------------------------------------------
 100010         5279  5  DM_NESTED_NUMERICALS(
```

```
                         DM_NESTED_NUMERICAL(
                          '18" Flat Panel Graphics Monitor', 1),
                         DM_NESTED_NUMERICAL(
                           'SIMM- 16MB PCMCIAII card', 1))
100011     5277.5    4  DM_NESTED_NUMERICALS(
                         DM_NESTED_NUMERICAL(
                           'External 8X CD-ROM', 1),
                         DM_NESTED_NUMERICAL(
                           'Mouse Pad', 100),
                         DM_NESTED_NUMERICAL(
                           'SIMM- 16MB PCMCIAII card', 1),
                         DM_NESTED_NUMERICAL(
                           'Keyboard Wrist Rest', 1),
                         DM_NESTED_NUMERICAL(
                           '18" Flat Panel Graphics Monitor', 1),
                         DM_NESTED_NUMERICAL(
                           'O/S Documentation Set - English', 1))
```

42.3 DBMS_PREDICTIVE_ANALYTICS

Machine learning can discover useful information buried in vast amounts of data. However, both the programming interfaces and the machine learning expertise required to obtain these results are too complex for use by the wide audiences that can obtain benefits from using Oracle Machine Learning for SQL.

The `DBMS_PREDICTIVE_ANALYTICS` package addresses both of these complexities by automating the entire machine learning process from data preprocessing through model building to scoring new data. This package provides an important tool that makes machine learning possible for a broad audience of users, in particular, business analysts.

This chapter contains the following topics:

- Overview
- Security Model
- Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms

42.3.1 Using DBMS_PREDICTIVE_ANALYTICS

This section contains topics that relate to using the `DBMS_PREDICTIVE_ANALYTICS` package.

- Overview
- Security Model

42.3.1.1 DBMS_PREDICTIVE_ANALYTICS Overview

DBMS_PREDICTIVE_ANALYTICS automates parts of the machine learning process.

Machine learning, according to a commonly used process model, requires the following steps:

1. Understand the business problem.
2. Understand the data.
3. Prepare the data for mining.
4. Create models using the prepared data.

5. Evaluate the models.
6. Deploy and use the model to score new data.

DBMS_PREDICTIVE_ANALYTICS automates parts of step 3 — 5 of this process.

Predictive analytics procedures analyze and prepare the input data, create and test machine learning models using the input data, and then use the input data for scoring. The results of scoring are returned to the user. The models and supporting objects are not preserved after the operation completes.

42.3.1.2 DBMS_PREDICTIVE_ANALYTICS Security Model

The DBMS_PREDICTIVE_ANALYTICS package is owned by user SYS and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The DBMS_PREDICTIVE_ANALYTICS package exposes APIs which are leveraged by the Oracle Machine Learning for SQL option. Users who wish to invoke procedures in this package require the CREATE MINING MODEL system privilege (as well as the CREATE TABLE and CREATE VIEW system privilege).

42.3.2 Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms

This table lists and briefly describes the DBMS_PREDICTIVE_ANALYTICS package subprograms.

Table 42-173 DBMS_PREDICTIVE_ANALYTICS Package Subprograms

Subprogram	Purpose
EXPLAIN Procedure	Ranks attributes in order of influence in explaining a target column.
PREDICT Procedure	Predicts the value of a target column based on values in the input data.
PROFILE Procedure	Generates rules that identify the records that have the same target value.

42.3.2.1 EXPLAIN Procedure

The EXPLAIN procedure identifies the attributes that are important in explaining the variation in values of a target column.

The input data must contain some records where the target value is known (not NULL). These records are used by the procedure to train a model that calculates the attribute importance.

Note:

EXPLAIN supports DATE and TIMESTAMP datatypes in addition to the numeric, character, and nested datatypes supported by Oracle Machine Learning for SQL models.

Data requirements for Oracle Machine Learning for SQL are described in *Oracle Machine Learning for SQL User's Guide*

The EXPLAIN procedure creates a result table that lists the attributes in order of their explanatory power. The result table is described in the Usage Notes.

Syntax

```
DBMS_PREDICTIVE_ANALYTICS.EXPLAIN (
     data_table_name     IN VARCHAR2,
     explain_column_name IN VARCHAR2,
     result_table_name   IN VARCHAR2,
     data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-174 EXPLAIN Procedure Parameters

Parameter	Description
data_table_name	Name of input table or view
explain_column_name	Name of the column to be explained
result_table_name	Name of the table where results are saved
data_schema_name	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

Usage Notes

The EXPLAIN procedure creates a result table with the columns described in Table 42-175.

Table 42-175 EXPLAIN Procedure Result Table

Column Name	Datatype	Description
ATTRIBUTE_NAME	VARCHAR2(30)	Name of a column in the input data; all columns except the explained column are listed in the result table.

Table 42-175 (Cont.) EXPLAIN Procedure Result Table

Column Name	Datatype	Description
`EXPLANATORY_VALUE`	`NUMBER`	Value indicating how useful the column is for determining the value of the explained column. Higher values indicate greater explanatory power. Value can range from 0 to 1. An individual column's explanatory value is independent of other columns in the input table. The values are based on how strong each individual column correlates with the explained column. The value is affected by the number of records in the input table, and the relations of the values of the column to the values of the explain column. An explanatory power value of 0 implies there is no useful correlation between the column's values and the explain column's values. An explanatory power of 1 implies perfect correlation; such columns should be eliminated from consideration for `PREDICT`. In practice, an explanatory power equal to 1 is rarely returned.
`RANK`	`NUMBER`	Ranking of explanatory power. Rows with equal values for `explanatory_value` have the same rank. Rank values are not skipped in the event of ties.

Example

The following example performs an `EXPLAIN` operation on the `SUPPLEMENTARY_DEMOGRAPHICS` table of Sales History.

```
--Perform EXPLAIN operation
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'supplementary_demographics',
        explain_column_name  => 'home_theater_package',
        result_table_name    => 'demographics_explain_result');
END;
/
--Display results
SELECT * FROM demographics_explain_result;

ATTRIBUTE_NAME                           EXPLANATORY_VALUE       RANK
---------------------------------------- ----------------- ----------
Y_BOX_GAMES                                     .524311073          1
YRS_RESIDENCE                                   .495987246          2
HOUSEHOLD_SIZE                                  .146208506          3
AFFINITY_CARD                                     .0598227          4
EDUCATION                                       .018462703          5
OCCUPATION                                      .009721543          6
FLAT_PANEL_MONITOR                               .00013733          7
PRINTER_SUPPLIES                                         0          8
OS_DOC_SET_KANJI                                         0          8
BULK_PACK_DISKETTES                                      0          8
BOOKKEEPING_APPLICATION                                  0          8
COMMENTS                                                 0          8
CUST_ID                                                  0          8
```

The results show that `Y_BOX_GAMES`, `YRS_RESIDENCE`, and `HOUSEHOLD_SIZE` are the best predictors of `HOME_THEATER_PACKAGE`.

42.3.2.2 PREDICT Procedure

The PREDICT procedure predicts the values of a target column.

The input data must contain some records where the target value is known (not NULL). These records are used by the procedure to train and test a model that makes the predictions.

> **Note:**
>
> PREDICT supports DATE and TIMESTAMP datatypes in addition to the numeric, character, and nested datatypes supported by Oracle Machine Learning for SQL models.
>
> Data requirements for OML4SQL are described in *Oracle Machine Learning for SQL User's Guide*

The PREDICT procedure creates a result table that contains a predicted target value for every record. The result table is described in the Usage Notes.

Syntax

```
DBMS_PREDICTIVE_ANALYTICS.PREDICT (
    accuracy                  OUT NUMBER,
    data_table_name           IN VARCHAR2,
    case_id_column_name       IN VARCHAR2,
    target_column_name        IN VARCHAR2,
    result_table_name         IN VARCHAR2,
    data_schema_name          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-176 PREDICT Procedure Parameters

Parameter	Description
accuracy	Output parameter that returns the predictive confidence, a measure of the accuracy of the predicted values. The predictive confidence for a categorical target is the most common target value; the predictive confidence for a numerical target is the mean.
data_table_name	Name of the input table or view.
case_id_column_name	Name of the column that uniquely identifies each case (record) in the input data.
target_column_name	Name of the column to predict.
result_table_name	Name of the table where results will be saved.
data_schema_name	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

Usage Notes

The PREDICT procedure creates a result table with the columns described in Table 42-177.

Table 42-177 PREDICT Procedure Result Table

Column Name	Datatype	Description
Case ID column name	`VARCHAR2` or `NUMBER`	The name of the case ID column in the input data.
`PREDICTION`	`VARCHAR2` or `NUMBER`	The predicted value of the target column for the given case.
`PROBABILITY`	`NUMBER`	For classification (categorical target), the probability of the prediction. For regression problems (numerical target), this column contains `NULL`.

> **Note:**
>
> Make sure that the name of the case ID column is not '`PREDICTION`' or '`PROBABILITY`'.

Predictions are returned for all cases whether or not they contained target values in the input.

Predicted values for known cases may be interesting in some situations. For example, you could perform deviation analysis to compare predicted values and actual values.

Example

The following example performs a `PREDICT` operation and displays the first 10 predictions. The results show an accuracy of 79% in predicting whether each customer has an affinity card.

```
--Perform PREDICT operation
DECLARE
    v_accuracy NUMBER(10,9);
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.PREDICT(
        accuracy             => v_accuracy,
        data_table_name      => 'supplementary_demographics',
        case_id_column_name  => 'cust_id',
        target_column_name   => 'affinity_card',
        result_table_name    => 'pa_demographics_predict_result');
    DBMS_OUTPUT.PUT_LINE('Accuracy = ' || v_accuracy);
END;
/

Accuracy = .788696903

--Display results
SELECT * FROM pa_demographics_predict_result WHERE rownum < 10;

   CUST_ID PREDICTION PROBABILITY
---------- ---------- -----------
    101501          1  .834069848
    101502          0  .991269965
    101503          0   .99978311
    101504          1  .971643388
    101505          1  .541754127
```

```
101506          0   .803719133
101507          0   .999999303
101508          0   .999999987
101509          0   .999953074
```

42.3.2.3 PROFILE Procedure

The `PROFILE` procedure generates rules that describe the cases (records) from the input data.

For example, if a target column `CHURN` has values 'Yes' and 'No', `PROFILE` generates a set of rules describing the expected outcomes. Each profile includes a rule, record count, and a score distribution.

The input data must contain some cases where the target value is known (not `NULL`). These cases are used by the procedure to build a model that calculates the rules.

> **Note:**
>
> `PROFILE` does not support nested types or dates.
>
> Data requirements for Oracle Machine Learning for SQL are described in *Oracle Machine Learning for SQL User's Guide*

The `PROFILE` procedure creates a result table that specifies rules (profiles) and their corresponding target values. The result table is described in the Usage Notes.

Syntax

```
DBMS_PREDICTIVE_ANALYTICS.PROFILE (
     data_table_name            IN VARCHAR2,
     target_column_name         IN VARCHAR2,
     result_table_name          IN VARCHAR2,
     data_schema_name           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 42-178   PROFILE Procedure Parameters

Parameter	Description
`data_table_name`	Name of the table containing the data to be analyzed.
`target_column_name`	Name of the target column.
`result_table_name`	Name of the table where the results will be saved.
`data_schema_name`	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

Usage Notes

The `PROFILE` procedure creates a result table with the columns described in Table 42-179.

Table 42-179 PROFILE Procedure Result Table

Column Name	Datatype	Description
PROFILE_ID	NUMBER	A unique identifier for this profile (rule).
RECORD_COUNT	NUMBER	The number of records described by the profile.
DESCRIPTION	SYS.XMLTYPE	The profile rule. See "XML Schema for Profile Rules".

XML Schema for Profile Rules

The DESCRIPTION column of the result table contains XML that conforms to the following XSD:

```
<xs:element name="SimpleRule">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="PREDICATE"/>
      <xs:element ref="ScoreDistribution" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="optional"/>
    <xs:attribute name="score" type="xs:string" use="required"/>
    <xs:attribute name="recordCount" type="NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Example

This example generates a rule describing customers who are likely to use an affinity card (target value is 1) and a set of rules describing customers who are not likely to use an affinity card (target value is 0). The rules are based on only two predictors: education and occupation.

```
SET serveroutput ON
SET trimspool ON
SET pages 10000
SET long 10000
SET pagesize 10000
SET linesize 150
CREATE VIEW cust_edu_occ_view AS
              SELECT cust_id, education, occupation, affinity_card
              FROM sh.supplementary_demographics;
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.PROFILE(
         DATA_TABLE_NAME    => 'cust_edu_occ_view',
         TARGET_COLUMN_NAME => 'affinity_card',
         RESULT_TABLE_NAME  => 'profile_result');
END;
/
```

This example generates eight rules in the result table profile_result. Seven of the rules suggest a target value of 0; one rule suggests a target value of 1. The score attribute on a rule identifies the target value.

This SELECT statement returns all the rules in the result table.

```
SELECT a.profile_id, a.record_count, a.description.getstringval()
 FROM profile_result a;
```

This SELECT statement returns the rules for a target value of 0.

```
SELECT *
  FROM profile_result t
  WHERE extractvalue(t.description, '/SimpleRule/@score') = 0;
```

The eight rules generated by this example are displayed as follows.

```
<SimpleRule id="1" score="0" recordCount="443">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"< Bach." "Assoc-V" "HS-grad"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="297" />
  <ScoreDistribution value="1" recordCount="146" />
</SimpleRule>

<SimpleRule id="2" score="0" recordCount="18">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "Presch."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="18" />
</SimpleRule>

<SimpleRule id="3" score="0" recordCount="458">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="248" />
  <ScoreDistribution value="1" recordCount="210" />
</SimpleRule>

<SimpleRule id="4" score="1" recordCount="276">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Masters" "PhD" "Profsc"
      </Array>
```

```
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="1" recordCount="183" />
  <ScoreDistribution value="0" recordCount="93" />
</SimpleRule>

<SimpleRule id="5" score="0" recordCount="307">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach." "Masters" "PhD" "Profsc"
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Crafts" "Sales" "TechSup" "Transp."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="184" />
  <ScoreDistribution value="1" recordCount="123" />
</SimpleRule>

<SimpleRule id="6" score="0" recordCount="243">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach." "Masters" "PhD" "Profsc"
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"?" "Cleric." "Farming" "Handler" "House-s" "Machine" "Other"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="197" />
  <ScoreDistribution value="1" recordCount="46" />
</SimpleRule>

<SimpleRule id="7" score="0" recordCount="2158">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">
        "10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "< Bach." "Assoc-V" "HS-grad"
        "Presch."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"?" "Cleric." "Crafts" "Farming" "Machine" "Sales" "TechSup" "
Transp."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="1819"/>
  <ScoreDistribution value="1" recordCount="339"/>
</SimpleRule>

<SimpleRule id="8" score="0" recordCount="597">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">
        "10th" "11th" "12th" "1st-4th"  "5th-6th" "7th-8th" "9th" "< Bach." "Assoc-V" "HS-grad"
        "Presch."
      </Array>
```

```
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Handler" "House-s" "Other"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="0" recordCount="572"/>
<ScoreDistribution value="1" recordCount="25"/>
</SimpleRule>
```

43
Data Dictionary Views

The information in the data dictionary tables can be viewed through data dictionary views. The Oracle Machine Learning for SQL related dictionary views are listed in this chapter.

- ALL_MINING_MODELS
- ALL_MINING_MODEL_ATTRIBUTES
- ALL_MINING_MODEL_PARTITIONS
- ALL_MINING_MODEL_SETTINGS
- ALL_MINING_MODEL_VIEWS
- ALL_MINING_MODEL_XFORMS

43.1 ALL_MINING_MODELS

`ALL_MINING_MODELS` describes the machine learning models accessible to the current user.

Mining models are schema objects created by Oracle Machine Learning for SQL.

Related Views

- `DBA_MINING_MODELS` describes all machine learning models in the database.
- `USER_MINING_MODELS` describes the machine learning models owned by the current user. This view does not display the `OWNER` column.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)	NOT NULL	Owner of the machine learning model
MODEL_NAME	VARCHAR2(128)	NOT NULL	Name of the machine learning model
MINING_FUNCTION	VARCHAR2(30)		Function of the mining model. The function identifies the class of problems that can be solved by this model. The machine learning function is specified when the model is built: • CLASSIFICATION • REGRESSION • CLUSTERING • EMBEDDING • FEATURE_EXTRACTION • ASSOCIATION_RULES • ATTRIBUTE_IMPORTANCE • TIME_SERIES

Column	Datatype	NULL	Description
ALGORITHM	VARCHAR2(30)		Algorithm used by the model. Each machine learning function has a default algorithm. The default can be overridden with a model setting (see *_MINING_MODEL_SETTINGS): • APRIORI_ASSOCIATION_RULES • CUR_DECOMPOSITION • DECISION_TREE • EXPECTATION_MAXIMIZATION • EXPLICIT_SEMANTIC_ANALYS • EXPONENTIAL_SMOOTHING • EXTENSIBLE_LANG • GENERALIZED_LINEAR_MODEL • KMEANS • MINIMUM_DESCRIPTION_LENGTH • MSET_SPRT • NAIVE_BAYES • NEURAL_NETWORK • NONNEGATIVE_MATRIX_FACTOR • O_CLUSTER • ONNX • RANDOM_FOREST • SUPPORT_VECTOR_MACHINE • SINGULAR_VALUE_DECOMP • XGBOOST
ALGORITHM_TYPE	VARCHAR2(10)		R type algorithm. This column is used in R algorithm registration.
CREATION_DATE	DATE	NOT NULL	Date that the model was created
BUILD_DURATION	NUMBER		Time (in seconds) of the model build process
MODEL_SIZE	NUMBER		Size of the model (in megabytes)
PARTITIONED	VARCHAR2(3)		Indicates whether the model is partitioned or not. Possible values: • YES: The model is partitioned. • NO: The model is not partitioned
BUILD_SOURCE	CLOB		Input data source (provided by the user at build time) on which to build the model This column is populated for models created in Oracle Database 23ai or later. For older version models that were imported into Oracle Database 23ai or later, the value of this column is null.
COMMENTS	VARCHAR2(4000)		Comment applied to the model with a SQL COMMENT statement

Related Topics

- *DBA_MINING_MODEL*
- *USER_MINING_MODELS*

43.2 ALL_MINING_MODEL_ATTRIBUTES

ALL_MINING_MODEL_ATTRIBUTES describes the attributes of the machine learning models accessible to the current user.

Only the attributes in the model signature are included in this view. The attributes in the model signature correspond to the columns in the training data that were used to build the model.

Machine learning models are schema objects created by Oracle Machine Learning for SQL.

Related Views

- DBA_MINING_MODEL_ATTRIBUTES describes the attributes of all machine learning models in the database.
- USER_MINING_MODEL_ATTRIBUTES describes the attributes of the machine learning models owned by the current user. This view does not display the OWNER column.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)	NOT NULL	Owner of the machine learning model
MODEL_NAME	VARCHAR2(128)	NOT NULL	Name of the machine learning model
ATTRIBUTE_NAME	VARCHAR2(128)	NOT NULL	Name of the attribute The target attribute name for ONNX models (mining model with ALGORITHM set to ONNX) is always ORA$ONNXTARGET.
ATTRIBUTE_TYPE	VARCHAR2(11)	–	Logical type of the attribute. The type is identified during the model build or apply process: • NUMERICAL: Numeric data • CATEGORICAL: Character data • TEXT: Unstructured text data • PARTITION: The input signature column is used for the partitioning key • MIXED: The input signature column takes on more than one attribute type. This is due to user-defined embedded transformations that allow an input column to be transformed into multiple independent mining attributes, including mining attributes of different types. • VECTOR: Attribute of type vectors (typically, target attribute of embedding models).
DATA_TYPE	VARCHAR2(106)	–	Data type of the attribute
DATA_LENGTH	NUMBER	–	Length of the data type
DATA_PRECISION	NUMBER	–	Precision of a fixed point number. Precision, which is the total number of significant decimal digits, is represented as *p* in the data type NUMBER(p,s).
DATA_SCALE	NUMBER	–	Scale of a fixed point number. Scale, which is the number of digits from the decimal to the least significant digit, is represented as *s* in the data type NUMBER(p,s).

Column	Datatype	NULL	Description
USAGE_TYPE	VARCHAR2(8)	–	Indicates whether the attribute was used to construct the model (ACTIVE) or not (INACTIVE). Some attributes may be eliminated by transformations or algorithmic processing. The *_MINING_MODEL_ATTRIBUTES view only lists the attributes used by the model, therefore the value of this column is always ACTIVE.
TARGET	VARCHAR2(3)	–	Indicates whether the attribute is the target of a predictive model (YES) or not (NO). The target describes the result that is produced when the model is applied.
ATTRIBUTE_SPEC	VARCHAR2(4000)	–	One or more keywords that identify special treatment for the attribute during model build. Values are: • FORCE_IN: (GLM only) When feature selection is enabled, forces the inclusion of the attribute in the model build. Feature selection is disabled by default. If the model is not using GLM with feature selection enabled, this value is ignored. • NOPREP: When ADP is on, prevents automatic transformation of the attribute. If ADP is OFF, this value is ignored. • TEXT: Causes the attribute to be treated as unstructured text data. The TEXT value supports three subsettings: POLICY_NAME, MAX_FEATURES, TOKEN_TYPE, and MIN_DOCUMENTS. Subsettings are specified as name:value pairs within parentheses. For example: (POLICY_NAME:mypolicy)(MAX_FEATURES:2000)(TOKEN_TYPE:THEME). See *Oracle Machine Learning for SQL API Guide* for details. • NULL: The ATTRIBUTE_SPEC for this attribute is NULL. ATTRIBUTE_SPEC is a parameter to the PL/SQL procedure DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM. See *Oracle Database PL/SQL Packages and Types Reference* for details.
VECTOR_INFO	VECTOR(<dimension>, <element_type>)	–	Indicates the number of vectors with their data type in an ONNX model. For example, VECTOR(768,float32).

Related Topics

- *DBA_MINING_MODEL_ATTRIBUTES*
- *USER_MINING_MODEL_ATTRIBUTES*

43.3 ALL_MINING_MODEL_PARTITIONS

ALL_MINING_MODEL_PARTITIONS describes all the model partitions accessible to the user.

Related Views

- DBA_MINING_MODEL_PARTITIONS describes all the model partitions accessible to the system.
- USER_MINING_MODEL_PARTITIONS describes the user's own model partitions. This view does not display the OWNER column.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)	NOT NULL	Name of the model owner
MODEL_NAME	VARCHAR2(128)	NOT NULL	Name of the model
PARTITION_NAME	VARCHAR2(128)	–	Name of the model partition
POSITION	NUMBER	–	Column position number for partitioning column. Column position represents the position of the column in a multi-column partitioning key, or 1 for a unary column partitioning key.
COLUMN_NAME	VARCHAR2(128)	NOT NULL	Name of the column used for partitioning
COLUMN_VALUE	VARCHAR2(4000)	–	Value of the column for this partition

Related Topics

- *DBA_MINING_MODEL_PARTITIONS*
- *USER_MINING_MODEL_PARTITIONS*

43.4 ALL_MINING_MODEL_SETTINGS

ALL_MINING_MODEL_SETTINGS describes the settings of the machine learning models accessible to the current user.

Machine learning models are schema objects created by Oracle Machine Learning for SQL.

Related Views

- DBA_MINING_MODEL_SETTINGS describes the settings of all machine learning models in the database.
- USER_MINING_MODEL_SETTINGS describes the settings of the machine learning models owned by the current user. This view does not display the OWNER column.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)	NOT NULL	Owner of the machine learning model
MODEL_NAME	VARCHAR2(128)	NOT NULL	Name of the machine learning model
SETTING_NAME	VARCHAR2(30)	NOT NULL	Name of the setting
SETTING_VALUE	VARCHAR2(4000)	–	Value of the setting

Column	Datatype	NULL	Description
`SETTING_TYPE`	`VARCHAR2(7)`	–	Indicates whether the default value (`DEFAULT`) or a user-specified value (`INPUT`) is used by the model

Related Topics

- *DBA_MINING_MODEL_SETTINGS*
- *USER_MINING_MODEL_SETTINGS*

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for descriptions of model settings

43.5 ALL_MINING_MODEL_VIEWS

`ALL_MINING_MODEL_VIEWS` provides a description of all the model views accessible to the user.

Related Views

- `DBA_MINING_MODEL_VIEWS` provides a description of all the model views in the database.
- `USER_MINING_MODEL_VIEWS` provides a description of the user's own model views. This view does not display the `OWNER` column.

Column	Datatype	NULL	Description
`OWNER`	`VARCHAR2(128)`	`NOT NULL`	Owner of the model view
`MODEL_NAME`	`VARCHAR2(128)`	`NOT NULL`	Name of the model to which model views belongs
`VIEW_NAME`	`VARCHAR2(128)`	`NOT NULL`	Name of the model view
`VIEW_TYPE`	`VARCHAR2(128)`	–	Type of the model view

Related Topics

- *DBA_MINING_MODEL_VIEWS*
- *USER_MINING_MODEL_VIEWS*

> **See Also:**
>
> "ALL_MINING_MODEL_VIEWS" in *Oracle Machine Learning for SQL User's Guide*

43.6 ALL_MINING_MODEL_XFORMS

ALL_MINING_MODEL_XFORMS describes the user-specified transformations embedded in all models accessible to the user.

Related Views

- DBA_MINING_MODEL_XFORMS describes the user-specified transformations embedded in all models accessible in the system.
- USER_MINING_MODEL_XFORMS describes the user-specified transformations embedded with the user's own models. This view does not display the OWNER column.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)	NOT NULL	Name of the model owner
MODEL_NAME	VARCHAR2(128)	NOT NULL	Name of the model
ATTRIBUTE_NAME	VARCHAR2(128)		Name of the attribute used in the transformation
ATTRIBUTE_SUBNAME	VARCHAR2(4000)		Subname of the attribute used in the transformation
ATTRIBUTE_SPEC	VARCHAR2(4000)		Attribute specification provided to model training
EXPRESSION	CLOB		Transformation expression provided to model training
REVERSE	VARCHAR2(3)		Indicates whether the specified transformation is a reverse transformation (YES) or a forward expression (NO)

Related Topics

- *DBA_MINING_MODEL_XFORMS*
- *USER_MINING_MODEL_XFORMS*

44

SQL Scoring Functions

Oracle Machine Learning for SQL functions are single-row functions that use OML4SQL to score data. The functions can apply a mining model schema object to the data, or they can dynamically mine the data by executing an analytic clause.

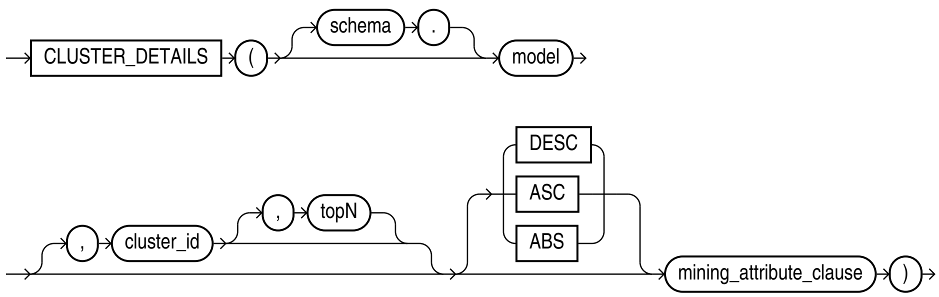
> **Note:**
>
> For a description of the syntax diagrams for these functions, see How to Read Syntax Diagrams in *Oracle Database SQL Language Reference*

- CLUSTER_DETAILS
- CLUSTER_DISTANCE
- CLUSTER_ID
- CLUSTER_PROBABILITY
- CLUSTER_SET
- FEATURE_COMPARE
- FEATURE_DETAILS
- FEATURE_ID
- FEATURE_SET
- FEATURE_VALUE
- ORA_DM_PARTITION_NAME
- PREDICTION
- PREDICTION_BOUNDS
- PREDICTION_COST
- PREDICTION_DETAILS
- PREDICTION_PROBABILITY
- PREDICTION_SET
- VECTOR_EMBEDDING

44.1 CLUSTER_DETAILS

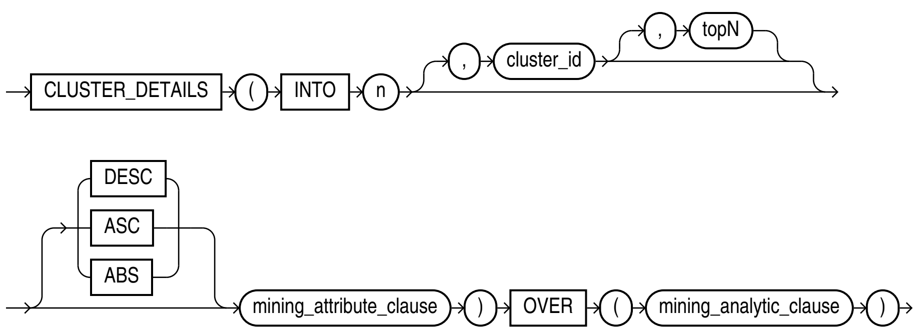
Syntax

***cluster_details*::=**

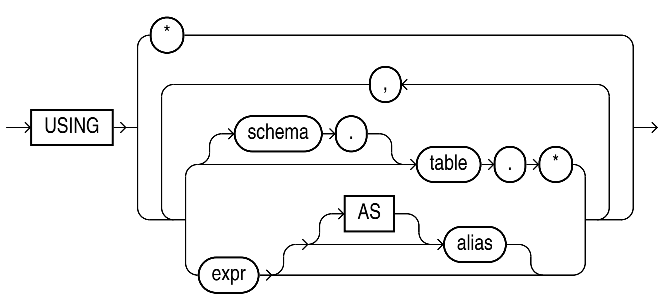


Analytic Syntax

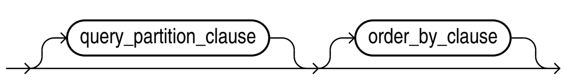
***cluster_details_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *mining_analytic_clause*

Purpose

CLUSTER_DETAILS returns cluster details for each row in the selection. The return value is an XML string that describes the attributes of the highest probability cluster or the specified *cluster_id*.

topN

If you specify a value for *topN*, the function returns the *N* attributes that most influence the cluster assignment (the score). If you do not specify *topN*, the function returns the 5 most influential attributes.

DESC, ASC, or ABS

The returned attributes are ordered by weight. The weight of an attribute expresses its positive or negative impact on cluster assignment. A positive weight indicates an increased likelihood of assignment. A negative weight indicates a decreased likelihood of assignment.

By default, CLUSTER_DETAILS returns the attributes with the highest positive weights (DESC). If you specify ASC, the attributes with the highest negative weights are returned. If you specify ABS, the attributes with the greatest weights, whether negative or positive, are returned. The results are ordered by absolute value from highest to lowest. Attributes with a zero weight are not included in the output.

Syntax Choice

CLUSTER_DETAILS can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a clustering model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include INTO *n*, where *n* is the number of clusters to compute, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the CLUSTER_DETAILS function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring.
- *Oracle Machine Learning for SQL Concepts* for information about clustering.

Note:

The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

This example lists the attributes that have the greatest impact (more that 20% probability) on cluster assignment for customer ID 100955. The query invokes the `CLUSTER_DETAILS` and `CLUSTER_SET` functions, which apply the clustering model `em_sh_clus_sample`.

```
SELECT S.cluster_id, probability prob,
       CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 USING T.*) det
FROM
  (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
    FROM mining_data_apply_v v
   WHERE cust_id = 100955) T,
  TABLE(T.pset) S
ORDER BY 2 DESC;

CLUSTER_ID  PROB DET
---------- -----
--------------------------------------------------------------------------------
        14 .6761 <Details algorithm="Expectation Maximization" cluster="14">
                 <Attribute name="AGE" actualValue="51" weight=".676" rank="1"/>
                 <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".557" rank="2"/>
                 <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".412" rank="3"/>
                 <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".171" rank="4"/>
                 <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight="-.003"rank="5"/>
                 </Details>

         3 .3227 <Details algorithm="Expectation Maximization" cluster="3">
                 <Attribute name="YRS_RESIDENCE" actualValue="3" weight=".323" rank="1"/>
                 <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".265" rank="2"/>
                 <Attribute name="EDUCATION" actualValue="HS-grad" weight=".172" rank="3"/>
                 <Attribute name="AFFINITY_CARD" actualValue="0" weight=".125" rank="4"/>
                 <Attribute name="OCCUPATION" actualValue="Crafts" weight=".055" rank="5"/>
                 </Details>
```

Analytic Example

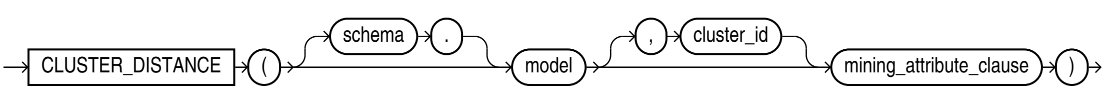
This example divides the customer database into four segments based on common characteristics. The clustering functions compute the clusters and return the score without a predefined clustering model.

```
SELECT * FROM (
     SELECT cust_id,
          CLUSTER_ID(INTO 4 USING *) OVER () cls,
          CLUSTER_DETAILS(INTO 4 USING *) OVER () cls_details
     FROM mining_data_apply_v)
WHERE cust_id <= 100003
ORDER BY 1;

CUST_ID CLS CLS_DETAILS
------- --- ------------------------------------------------------------------------------------
 100001   5 <Details algorithm="K-Means Clustering" cluster="5">
            <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".349" rank="1"/>
            <Attribute name="BULK_PACK_DISKETTES" actualValue="0" weight=".33" rank="2"/>
            <Attribute name="CUST_INCOME_LEVEL" actualValue="G: 130\,000 - 149\,999" weight=".291"
             rank="3"/>
            <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".268" rank="4"/>
            <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".179" rank="5"/>
            </Details>

 100002   6 <Details algorithm="K-Means Clustering" cluster="6">
            <Attribute name="CUST_GENDER" actualValue="F" weight=".945" rank="1"/>
            <Attribute name="CUST_MARITAL_STATUS" actualValue="NeverM" weight=".856" rank="2"/>
            <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".468" rank="3"/>
            <Attribute name="AFFINITY_CARD" actualValue="0" weight=".012" rank="4"/>
            <Attribute name="CUST_INCOME_LEVEL" actualValue="L: 300\,000 and above" weight=".009"
             rank="5"/>
            </Details>

 100003   7 <Details algorithm="K-Means Clustering" cluster="7">
            <Attribute name="CUST_MARITAL_STATUS" actualValue="NeverM" weight=".862" rank="1"/>
            <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".423" rank="2"/>
            <Attribute name="HOME_THEATER_PACKAGE" actualValue="0" weight=".113" rank="3"/>
            <Attribute name="AFFINITY_CARD" actualValue="0" weight=".007" rank="4"/>
            <Attribute name="CUST_ID" actualValue="100003" weight=".006" rank="5"/>
            </Details>
```

44.2 CLUSTER_DISTANCE

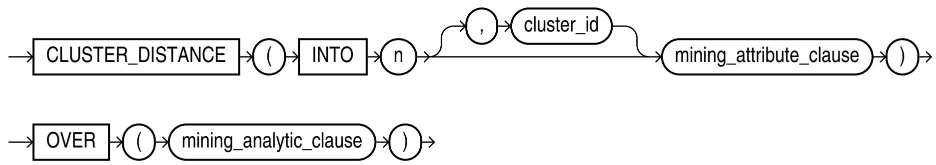
Syntax

***cluster_distance*::=**

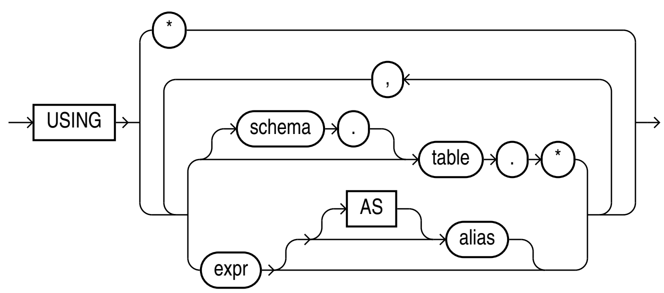


Analytic Syntax

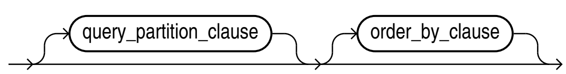
***cluster_distance_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of `mining_analytic_clause`

Purpose

`CLUSTER_DISTANCE` returns a cluster distance for each row in the selection. The cluster distance is the distance between the row and the centroid of the highest probability cluster or the specified *`cluster_id`*. The distance is returned as `BINARY_DOUBLE`.

Syntax Choice

`CLUSTER_DISTANCE` can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a clustering model.

- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include `INTO` *n*, where *n* is the number of clusters to compute, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the `CLUSTER_DISTANCE` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, this data is also used for building the transient models. The *mining_attribute_clause* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about clustering.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

This example finds the 10 rows that are most anomalous as measured by their distance from their nearest cluster centroid.

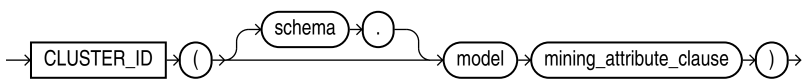
```
SELECT cust_id
  FROM (
    SELECT cust_id,
           rank() over
             (order by CLUSTER_DISTANCE(km_sh_clus_sample USING *) desc) rnk
      FROM mining_data_apply_v)
  WHERE rnk <= 11
  ORDER BY rnk;

   CUST_ID
----------
    100579
    100050
    100329
    100962
    101251
    100179
    100382
    100713
    100629
```

```
100787
101478
```

44.3 CLUSTER_ID

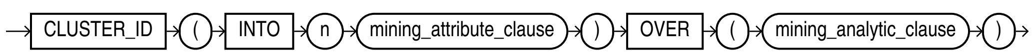
Syntax

***cluster_id*::=**

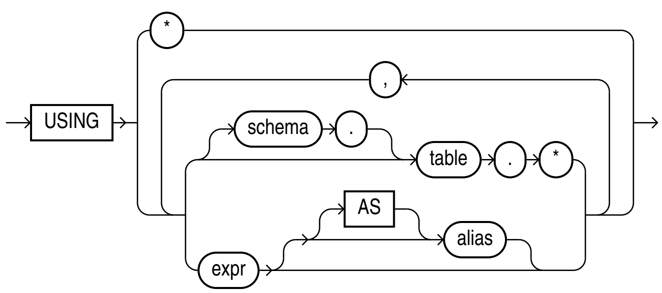


Analytic Syntax

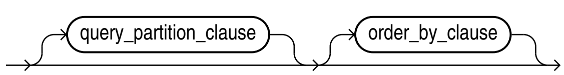
***cluster_id_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of `mining_analytic_clause`

Purpose

`CLUSTER_ID` returns the identifier of the highest probability cluster for each row in the selection. The cluster identifier is returned as an Oracle `NUMBER`.

Syntax Choice

`CLUSTER_ID` can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a clustering model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include `INTO` *`n`*, where *`n`* is the number of clusters to compute, and *`mining_analytic_clause`*, which specifies if the data should be partitioned for multiple model builds. The *`mining_analytic_clause`* supports a *`query_partition_clause`* and an *`order_by_clause`*. (See "analytic_clause::=".)

The syntax of the `CLUSTER_ID` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

`mining_attribute_clause` identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about clustering.

> **Note:**
>
> The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

The following example lists the clusters into which the customers in `mining_data_apply_v` have been grouped.

```
SELECT CLUSTER_ID(km_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
  FROM mining_data_apply_v
  GROUP BY CLUSTER_ID(km_sh_clus_sample USING *)
  ORDER BY cnt DESC;
```

```
      CLUS        CNT
---------- ----------
         2        580
        10        216
         6        186
         8        115
        19        110
        12        101
        18         81
        16         39
        17         38
        14         34
```

Analytic Example

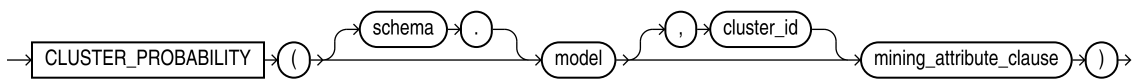
This example divides the customer database into four segments based on common characteristics. The clustering functions compute the clusters and return the score without a predefined clustering model.

```
SELECT * FROM (
     SELECT cust_id,
          CLUSTER_ID(INTO 4 USING *) OVER () cls,
          CLUSTER_DETAILS(INTO 4 USING *) OVER () cls_details
     FROM mining_data_apply_v)
WHERE cust_id <= 100003
ORDER BY 1;

CUST_ID CLS CLS_DETAILS
------- --- ------------------------------------------------------------------------------
 100001   5 <Details algorithm="K-Means Clustering" cluster="5">
            <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".349" rank="1"/>
            <Attribute name="BULK_PACK_DISKETTES" actualValue="0" weight=".33" rank="2"/>
            <Attribute name="CUST_INCOME_LEVEL" actualValue="G: 130\,000 - 149\,999"
               weight=".291" rank="3"/>
            <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".268" rank="4"/>
            <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".179" rank="5"/>
            </Details>

 100002   6 <Details algorithm="K-Means Clustering" cluster="6">
            <Attribute name="CUST_GENDER" actualValue="F" weight=".945" rank="1"/>
            <Attribute name="CUST_MARITAL_STATUS" actualValue="NeverM" weight=".856" rank="2"/>
            <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".468" rank="3"/>
            <Attribute name="AFFINITY_CARD" actualValue="0" weight=".012" rank="4"/>
            <Attribute name="CUST_INCOME_LEVEL" actualValue="L: 300\,000 and above"
               weight=".009" rank="5"/>
            </Details>

 100003   7 <Details algorithm="K-Means Clustering" cluster="7">
            <Attribute name="CUST_MARITAL_STATUS" actualValue="NeverM" weight=".862" rank="1"/>
            <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".423" rank="2"/>
            <Attribute name="HOME_THEATER_PACKAGE" actualValue="0" weight=".113" rank="3"/>
            <Attribute name="AFFINITY_CARD" actualValue="0" weight=".007" rank="4"/>
            <Attribute name="CUST_ID" actualValue="100003" weight=".006" rank="5"/>
            </Details>
```

44.4 CLUSTER_PROBABILITY

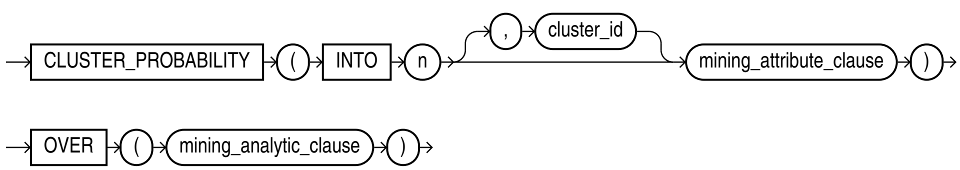
Syntax

***cluster_probability*::=**

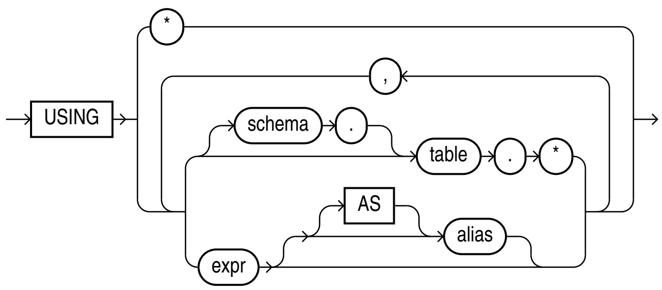


Analytic Syntax

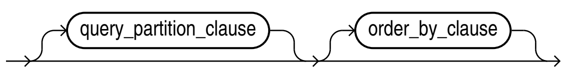
***cluster_prob_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



See Also:

"Analytic Functions" for information on the syntax, semantics, and restrictions of `mining_analytic_clause`

Purpose

CLUSTER_PROBABILITY returns a probability for each row in the selection. The probability refers to the highest probability cluster or to the specified *cluster_id*. The cluster probability is returned as BINARY_DOUBLE.

Syntax Choice

CLUSTER_PROBABILITY can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a clustering model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include INTO *n*, where *n* is the number of clusters to compute, and mining_analytic_clause, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the CLUSTER_PROBABILITY function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about clustering.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

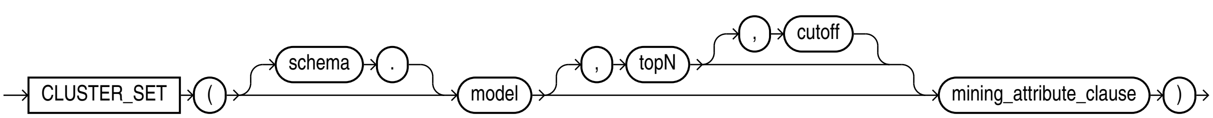
The following example lists the ten most representative customers, based on likelihood, of cluster 2.

```
SELECT cust_id
  FROM (SELECT cust_id, rank() OVER (ORDER BY prob DESC, cust_id) rnk_clus2
    FROM (SELECT cust_id, CLUSTER_PROBABILITY(km_sh_clus_sample, 2 USING *) prob
          FROM mining_data_apply_v))
WHERE rnk_clus2 <= 10
ORDER BY rnk_clus2;

   CUST_ID
----------
    100256
    100988
    100889
    101086
    101215
    100390
    100985
    101026
    100601
    100672
```

44.5 CLUSTER_SET

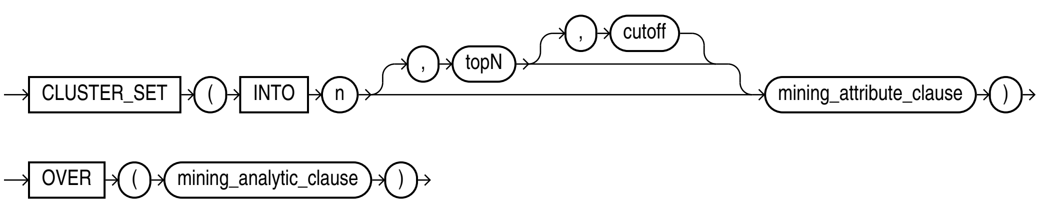
Syntax

***cluster_set*::=**

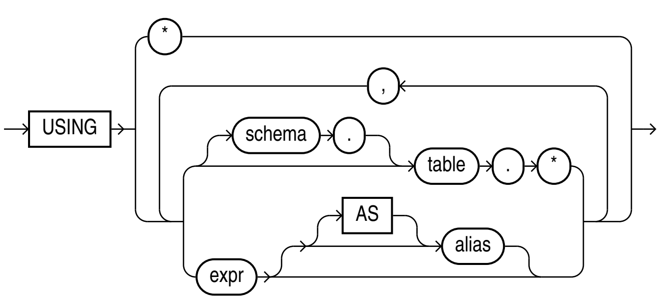


Analytic Syntax

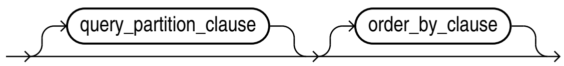
***cluster_set_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *mining_analytic_clause*

Purpose

`CLUSTER_SET` returns a set of cluster ID and probability pairs for each row in the selection. The return value is a varray of objects with field names `CLUSTER_ID` and `PROBABILITY`. The cluster identifier is an Oracle `NUMBER`; the probability is `BINARY_DOUBLE`.

topN and cutoff

You can specify *topN* and *cutoff* to limit the number of clusters returned by the function. By default, both *topN* and *cutoff* are null and all clusters are returned.

- *topN* is the *N* most probable clusters. If multiple clusters share the *N*th probability, then the function chooses one of them.
- *cutoff* is a probability threshold. Only clusters with probability greater than or equal to *cutoff* are returned. To filter by *cutoff* only, specify `NULL` for *topN*.

To return up to the *N* most probable clusters that are greater than or equal to *cutoff*, specify both *topN* and *cutoff*.

Syntax Choice

`CLUSTER_SET` can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a clustering model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include `INTO` *n*, where *n* is the number of clusters to compute, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the `CLUSTER_SET` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about clustering.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

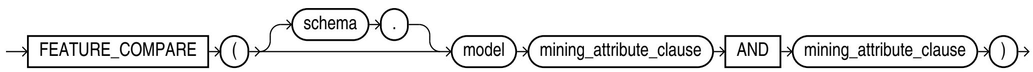
This example lists the attributes that have the greatest impact (more that 20% probability) on cluster assignment for customer ID 100955. The query invokes the CLUSTER_DETAILS and CLUSTER_SET functions, which apply the clustering model em_sh_clus_sample.

```
SELECT S.cluster_id, probability prob,
       CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 USING T.*) det
FROM
  (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
    FROM mining_data_apply_v v
   WHERE cust_id = 100955) T,
  TABLE(T.pset) S
ORDER BY 2 DESC;

CLUSTER_ID  PROB DET
---------- ----- ------------------------------------------------------------------------------
        14 .6761 <Details algorithm="Expectation Maximization" cluster="14">
                 <Attribute name="AGE" actualValue="51" weight=".676" rank="1"/>
                 <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".557" rank="2"/>
                 <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".412" rank="3"/>
                 <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".171" rank="4"/>
                 <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1" weight="-.003"rank="5"/>
                 </Details>

         3 .3227 <Details algorithm="Expectation Maximization" cluster="3">
                 <Attribute name="YRS_RESIDENCE" actualValue="3" weight=".323" rank="1"/>
                 <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".265" rank="2"/>
                 <Attribute name="EDUCATION" actualValue="HS-grad" weight=".172" rank="3"/>
                 <Attribute name="AFFINITY_CARD" actualValue="0" weight=".125" rank="4"/>
                 <Attribute name="OCCUPATION" actualValue="Crafts" weight=".055" rank="5"/>
                 </Details>
```

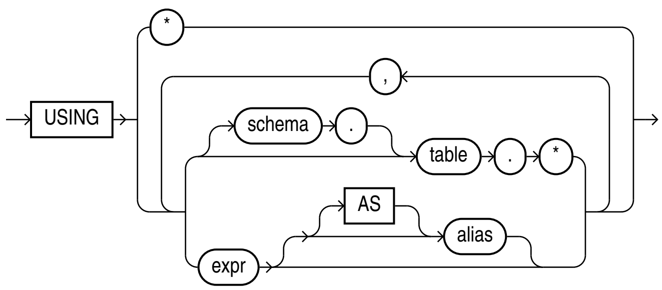
44.6 FEATURE_COMPARE

Syntax

***feature_compare*::=**



***mining_attribute_clause*::=**



Purpose

The `FEATURE_COMPARE` function uses a feature extraction model to compare two different documents, including short ones such as keyword phrases or two attribute lists, for similarity or dissimilarity. The `FEATURE_COMPARE` function can be used with Feature Extraction algorithms such as Singular Value Decomposition (SVD), Principal Component Analysis PCA), Non-negative Matrix Factorization (NMF), and Explicit Semantic Analysis (ESA). This function is applicable not only to documents, but also to numeric and categorical data.

The input to the `FEATURE_COMPARE` function is a single feature model built using the Feature Extraction algorithms of Oracle Machine Learning for SQL, such as NMF, SVD, and ESA. The double `USING` clause provides a mechanism to compare two different documents or constant keyword phrases, or any combination of the two, for similarity or dissimilarity using the extracted features in the model.

The syntax of the `FEATURE_COMPARE` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

The *`mining_attribute_clause`* identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. See *mining_attribute_clause*.

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring
- *Oracle Machine Learning for SQL Concepts* for information about clustering

Note:

The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Examples

An ESA model is built against a 2005 Wiki data set rendering over 200,000 features. The documents are mined as text and the document titles are considered as the Feature IDs.

The examples show the `FEATURE_COMPARE` function with the ESA algorithm, which compares a similar set of texts and then a dissimilar set of texts.

Similar texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA tour golfers from South
Africa' text AND USING 'Nick Price won the 2002 Mastercard Colonial Open' text) similarity
FROM DUAL;

SIMILARITY
----------
      .258
```

The output metric shows the results of a distance calculation. Therefore, a smaller number represents more similar texts. So 1 minus the distance in the queries represents a document similarity metric.

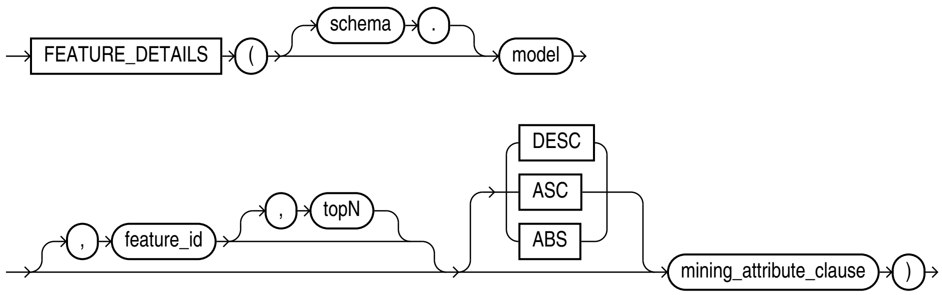
Dissimilar texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA tour golfers from South
Africa' text AND USING 'John Elway played quarterback for the Denver Broncos' text)
similarity FROM DUAL;

SIMILARITY
----------
      .007
```

44.7 FEATURE_DETAILS

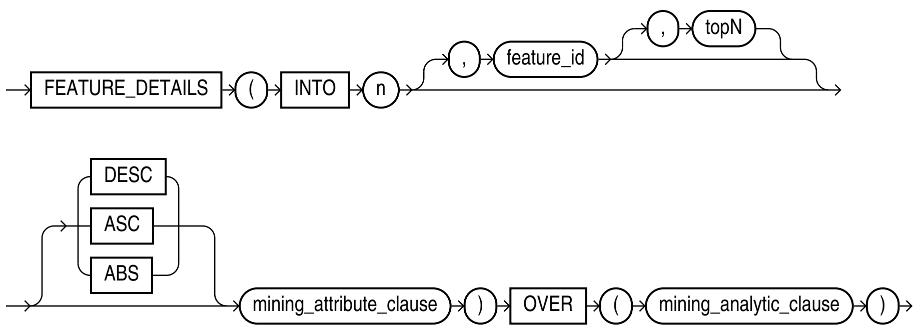
Syntax

***feature_details*::=**

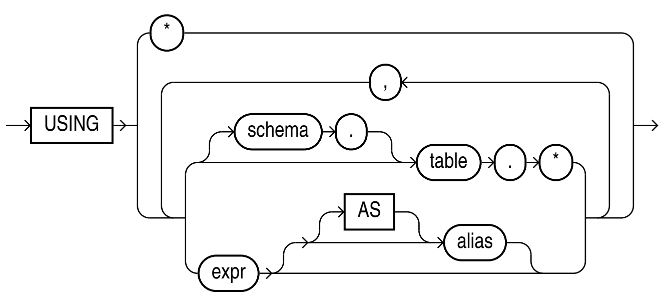


Analytic Syntax

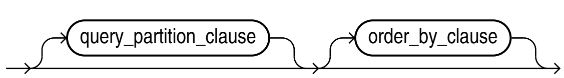
***feature_details_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *mining_analytic_clause*

Purpose

FEATURE_DETAILS returns feature details for each row in the selection. The return value is an XML string that describes the attributes of the highest value feature or the specified *feature_id*.

topN

If you specify a value for *topN*, the function returns the *N* attributes that most influence the feature value. If you do not specify *topN*, the function returns the 5 most influential attributes.

DESC, ASC, or ABS

The returned attributes are ordered by weight. The weight of an attribute expresses its positive or negative impact on the value of the feature. A positive weight indicates a higher feature value. A negative weight indicates a lower feature value.

By default, FEATURE_DETAILS returns the attributes with the highest positive weight (DESC). If you specify ASC, the attributes with the highest negative weight are returned. If you specify ABS, the attributes with the greatest weight, whether negative or positive, are returned. The results are ordered by absolute value from highest to lowest. Attributes with a zero weight are not included in the output.

Syntax Choice

FEATURE_DETAILS can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a feature extraction model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include INTO *n*, where *n* is the number of features to extract, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the FEATURE_DETAILS function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring.
- *Oracle Machine Learning for SQL Concepts* for information about feature extraction.

Note:

The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

This example uses the feature extraction model `nmf_sh_sample` to score the data. The query returns the three features that best represent customer 100002 and the attributes that most affect those features.

```
SELECT S.feature_id fid, value val,
       FEATURE_DETAILS(nmf_sh_sample, S.feature_id, 5 using T.*) det
   FROM
     (SELECT v.*, FEATURE_SET(nmf_sh_sample, 3 USING *) fset
         FROM mining_data_apply_v v
         WHERE cust_id = 100002) T,
   TABLE(T.fset) S
ORDER BY 2 DESC;

 FID    VAL  DET
---- ------  ----------------------------------------------------------------------------------
   5  3.492  <Details algorithm="Non-Negative Matrix Factorization" feature="5">
             <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".077" rank="1"/>
             <Attribute name="OCCUPATION" actualValue="Prof." weight=".062" rank="2"/>
             <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1" weight=".001" rank="3"/>
             <Attribute name="OS_DOC_SET_KANJI" actualValue="0" weight="0" rank="4"/>
             <Attribute name="YRS_RESIDENCE" actualValue="4" weight="0" rank="5"/>
             </Details>
   3  1.928  <Details algorithm="Non-Negative Matrix Factorization" feature="3">
             <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".239" rank="1"/>
             <Attribute name="CUST_INCOME_LEVEL" actualValue="L: 300\,000 and above"
              weight=".051" rank="2"/>
             <Attribute name="FLAT_PANEL_MONITOR" actualValue="1" weight=".02" rank="3"/>
             <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".006" rank="4"/>
             <Attribute name="AGE" actualValue="41" weight=".004" rank="5"/>
             </Details>
   8   .816  <Details algorithm="Non-Negative Matrix Factorization" feature="8">
             <Attribute name="EDUCATION" actualValue="Bach." weight=".211" rank="1"/>
             <Attribute name="CUST_MARITAL_STATUS" actualValue="NeverM" weight=".143" rank="2"/>
             <Attribute name="FLAT_PANEL_MONITOR" actualValue="1" weight=".137" rank="3"/>
             <Attribute name="CUST_GENDER" actualValue="F" weight=".044" rank="4"/>
             <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".032" rank="5"/>
             </Details>
```

Analytic Example

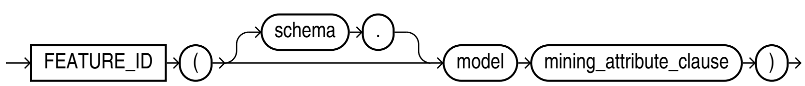
This example dynamically maps customer attributes into six features and returns the feature mapping for customer 100001.

```
SELECT feature_id, value
  FROM (
     SELECT cust_id, feature_set(INTO 6 USING *) OVER () fset
        FROM mining_data_apply_v),
  TABLE (fset)
  WHERE cust_id = 100001
  ORDER BY feature_id;

FEATURE_ID    VALUE
---------- --------
         1    2.670
         2     .000
         3    1.792
         4     .000
         5     .000
         6    3.379
```

44.8 FEATURE_ID

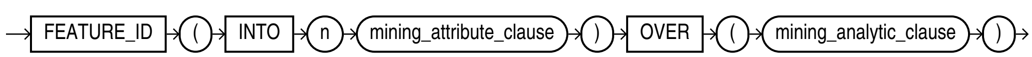
Syntax

***feature_id*::=**

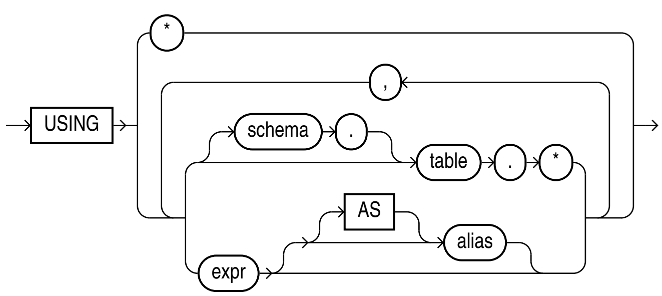


Analytic Syntax

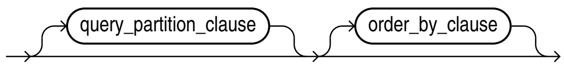
***feature_id_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *`mining_analytic_clause`*

Purpose

`FEATURE_ID` returns the identifier of the highest value feature for each row in the selection. The feature identifier is returned as an Oracle `NUMBER`.

Syntax Choice

`FEATURE_ID` can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a feature extraction model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include `INTO` *`n`*, where *`n`* is the number of features to extract, and *`mining_analytic_clause`*, which specifies if the data should be partitioned for multiple model builds. The *`mining_analytic_clause`* supports a *`query_partition_clause`* and an *`order_by_clause`*. (See "analytic_clause::=".)

The syntax of the `FEATURE_ID` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

`mining_attribute_clause` identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about feature extraction.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

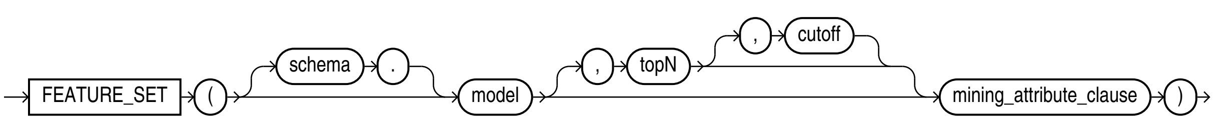
This example lists the features and corresponding count of customers in a data set.

```
SELECT FEATURE_ID(nmf_sh_sample USING *) AS feat, COUNT(*) AS cnt
  FROM nmf_sh_sample_apply_prepared
  GROUP BY FEATURE_ID(nmf_sh_sample USING *)
  ORDER BY cnt DESC, feat DESC;

      FEAT        CNT
---------- ----------
         7       1443
         2         49
         3          6
         6          1
         1          1
```

44.9 FEATURE_SET

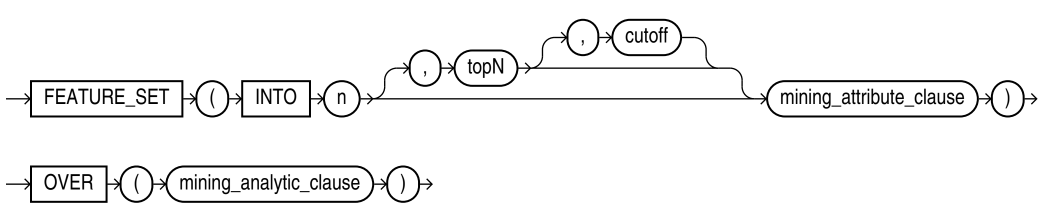
Syntax

***feature_set*::=**

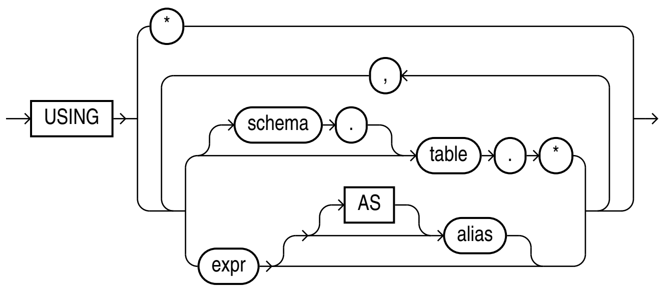


Analytic Syntax

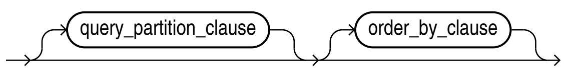
***feature_set_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of `mining_analytic_clause`

Purpose

`FEATURE_SET` returns a set of feature ID and feature value pairs for each row in the selection. The return value is a varray of objects with field names `FEATURE_ID` and `VALUE`. The data type of both fields is `NUMBER`.

topN and cutoff

You can specify *`topN`* and *`cutoff`* to limit the number of features returned by the function. By default, both *`topN`* and *`cutoff`* are null and all features are returned.

- *`topN`* is the *`N`* highest value features. If multiple features have the *`N`*th value, then the function chooses one of them.
- *`cutoff`* is a value threshold. Only features that are greater than or equal to *`cutoff`* are returned. To filter by *`cutoff`* only, specify `NULL` for *`topN`*.

To return up to *`N`* features that are greater than or equal to *`cutoff`*, specify both *`topN`* and *`cutoff`*.

Syntax Choice

`FEATURE_SET` can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a feature extraction model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include `INTO` *n*, where *n* is the number of features to extract, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the `FEATURE_SET` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about feature extraction.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

This example lists the top features corresponding to a given customer record and determines the top attributes for each feature (based on coefficient > 0.25).

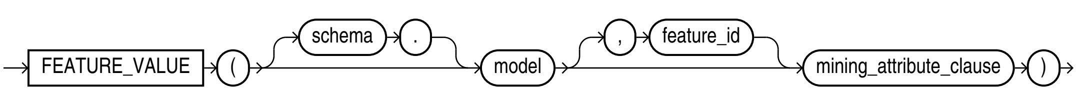
```
WITH
feat_tab AS (
SELECT F.feature_id fid,
       A.attribute_name attr,
       TO_CHAR(A.attribute_value) val,
       A.coefficient coeff
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF('nmf_sh_sample')) F,
       TABLE(F.attribute_set) A
 WHERE A.coefficient > 0.25
),
feat AS (
SELECT fid,
       CAST(COLLECT(Featattr(attr, val, coeff))
         AS Featattrs) f_attrs
  FROM feat_tab
GROUP BY fid
```

```
),
cust_10_features AS (
SELECT T.cust_id, S.feature_id, S.value
  FROM (SELECT cust_id, FEATURE_SET(nmf_sh_sample, 10 USING *) pset
          FROM nmf_sh_sample_apply_prepared
         WHERE cust_id = 100002) T,
       TABLE(T.pset) S
)
SELECT A.value, A.feature_id fid,
       B.attr, B.val, B.coeff
  FROM cust_10_features A,
       (SELECT T.fid, F.*
          FROM feat T,
               TABLE(T.f_attrs) F) B
 WHERE A.feature_id = B.fid
ORDER BY A.value DESC, A.feature_id ASC, coeff DESC, attr ASC, val ASC;

   VALUE  FID ATTR                      VAL                        COEFF
-------- ---- ------------------------- ------------------------ -------
  6.8409    7 YRS_RESIDENCE                                       1.3879
  6.8409    7 BOOKKEEPING_APPLICATION                              .4388
  6.8409    7 CUST_GENDER               M                          .2956
  6.8409    7 COUNTRY_NAME              United States of America   .2848
  6.4975    3 YRS_RESIDENCE                                       1.2668
  6.4975    3 BOOKKEEPING_APPLICATION                              .3465
  6.4975    3 COUNTRY_NAME              United States of America   .2927
  6.4886    2 YRS_RESIDENCE                                       1.3285
  6.4886    2 CUST_GENDER               M                          .2819
  6.4886    2 PRINTER_SUPPLIES                                     .2704
  6.3953    4 YRS_RESIDENCE                                       1.2931
  5.9640    6 YRS_RESIDENCE                                       1.1585
  5.9640    6 HOME_THEATER_PACKAGE                                 .2576
  5.2424    5 YRS_RESIDENCE                                       1.0067
  2.4714    8 YRS_RESIDENCE                                        .3297
  2.3559    1 YRS_RESIDENCE                                        .2768
  2.3559    1 FLAT_PANEL_MONITOR                                   .2593
```

44.10 FEATURE_VALUE

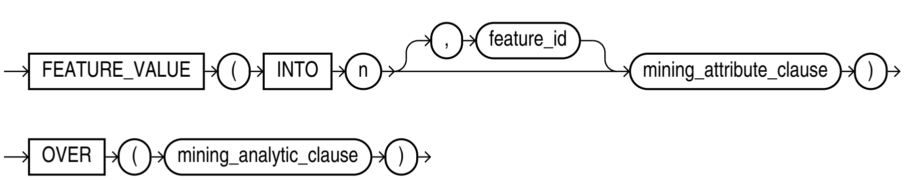
Syntax

***feature_value*::=**

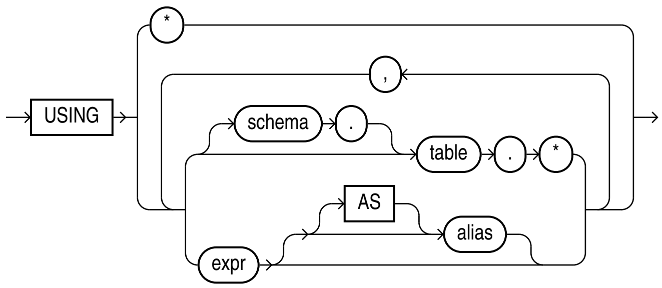


Analytic Syntax

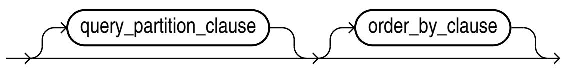
***feature_value_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *mining_analytic_clause*

Purpose

FEATURE_VALUE returns a feature value for each row in the selection. The value refers to the highest value feature or to the specified *feature_id*. The feature value is returned as BINARY_DOUBLE.

Syntax Choice

FEATURE_VALUE can score the data in one of two ways: It can apply a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax** — Use the first syntax to score the data with a pre-defined model. Supply the name of a feature extraction model.
- **Analytic Syntax** — Use the analytic syntax to score the data without a pre-defined model. Include INTO *n*, where *n* is the number of features to extract, and *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

The syntax of the FEATURE_VALUE function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, this data is also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring.
- *Oracle Machine Learning for SQL Concepts* for information about feature extraction.

Note:

The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

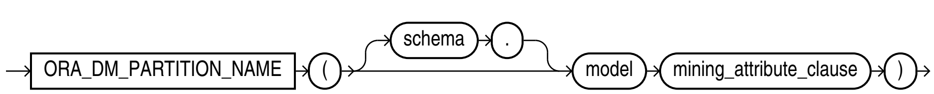
Example

The following example lists the customers that correspond to feature 3, ordered by match quality.

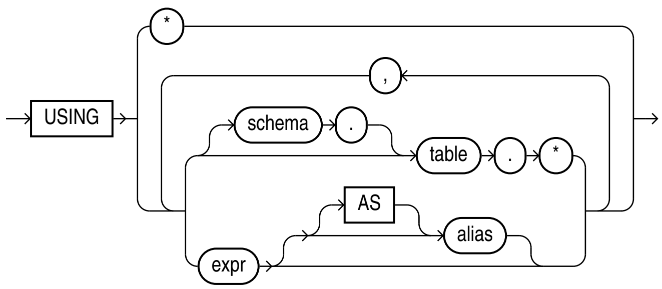
```
SELECT *
  FROM (SELECT cust_id, FEATURE_VALUE(nmf_sh_sample, 3 USING *) match_quality
          FROM nmf_sh_sample_apply_prepared
          ORDER BY match_quality DESC)
  WHERE ROWNUM < 11;

   CUST_ID MATCH_QUALITY
---------- -------------
    100210    19.4101627
    100962    15.2482251
    101151    14.5685197
    101499    14.4186292
    100363    14.4037396
    100372    14.3335148
    100982    14.1716545
    101039    14.1079914
    100759    14.0913761
    100953    14.0799737
```

44.11 ORA_DM_PARTITION_NAME

Syntax



***mining_attribute_clause*::=**



Purpose

`ORA_DM_PARTITION_NAME` is a single row function that works along with other existing functions. This function returns the name of the partition associated with the input row. When `ORA_DM_PARTITION_NAME` is used on a non-partitioned model, the result is `NULL`.

The syntax of the `ORA_DM_PARTITION_NAME` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

The *`mining_attribute_clause`* identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. See *mining_attribute_clause*.

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring
> - *Oracle Machine Learning for SQL Concepts* for information about clustering

> **Note:**
>
> The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

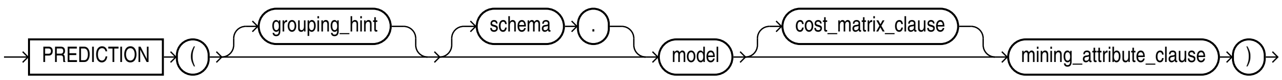
Example

```
SELECT prediction(mymodel using *) pred, ora_dm_partition_name(mymodel USING
*) pname FROM customers;
```

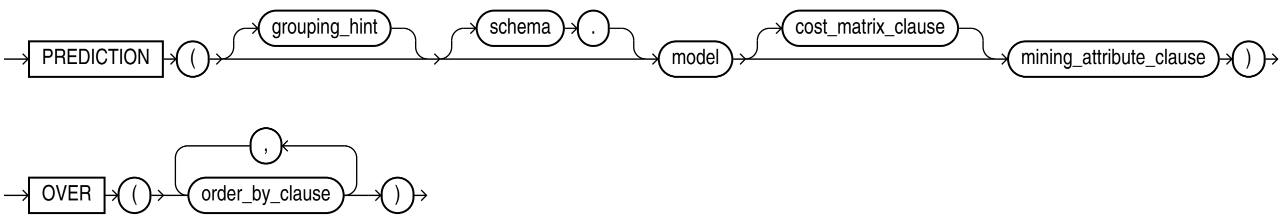
44.12 PREDICTION

Syntax

***prediction*::=**

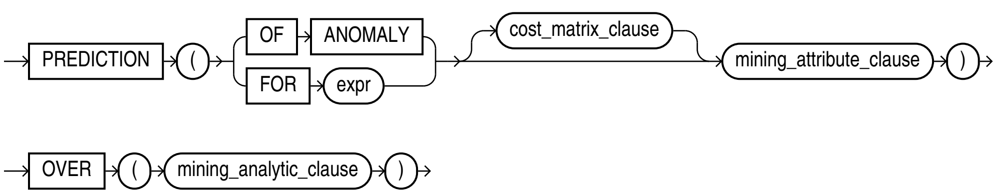


***prediction_ordered*::=**

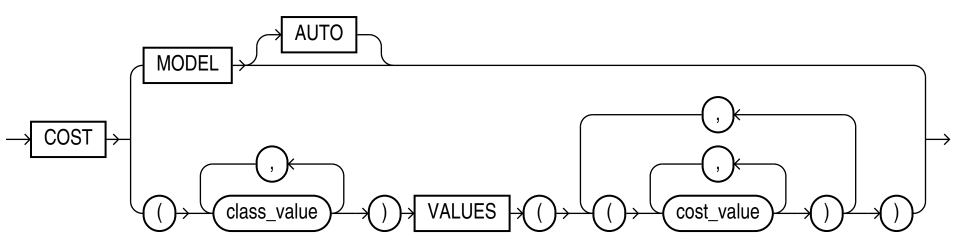


Analytic Syntax

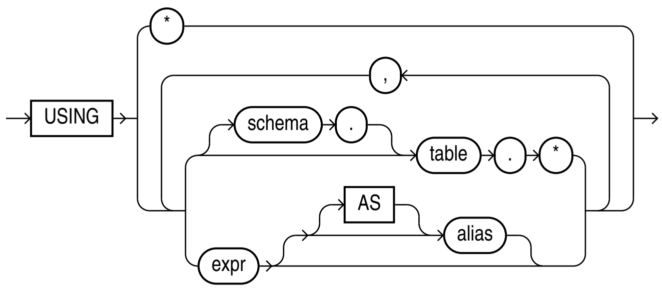
***prediction_analytic*::=**



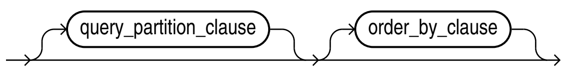
***cost_matrix_clause*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" in *Oracle Database SQL Language Reference* for information on the syntax, semantics, and restrictions of *`query_partition_clause`* and *`order_by_clause`*

Purpose

`PREDICTION` returns a prediction for each row in the selection. The data type of the returned prediction depends on whether the function performs Regression, Classification, or Anomaly Detection.

- **Regression**: Returns the expected target value for each row. The data type of the return value is the data type of the target.
- **Classification**: Returns the most probable target class (or lowest cost target class, if costs are specified) for each row. The data type of the return value is the data type of the target.
- **Anomaly Detection**: Returns `1` or `0` for each row. Typical rows are classified as 1. Rows that differ significantly from the rest of the data are classified as 0.

cost_matrix_clause

Costs are a biasing factor for minimizing the most harmful kinds of misclassifications. You can specify a *`cost_matrix_clause`* for classification or anomaly detection. Costs are not relevant for regression. The *`cost_matrix_clause`* behaves as described for "PREDICTION_COST".

Syntax Choice

`PREDICTION` can score data by applying a mining model object to the data, or it can dynamically score the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax**: Use the *prediction* syntax to score the data with a pre-defined model. Supply the name of a model that performs Classification, Regression, or Anomaly Detection.

  Use the *prediction_ordered* syntax for a model that requires ordered data, such as an MSET-SPRT model. The *prediction_ordered* syntax requires an *order_by_clause* clause.

  Restrictions on the *prediction_ordered* syntax are that you cannot use it in the `WHERE` clause of a query. Also, you cannot use a *query_partition_clause* or a *windowing_clause* with the *prediction_ordered* syntax.

  For details about the *order_by_clause*, see "Analytic Functions" in *Oracle Database SQL Language Reference*.

- **Analytic Syntax**: Use the *prediction_analytic* syntax to score the data without a pre-defined model. The analytic syntax uses the *mining_analytic_clause*, which specifies whether the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See the *analytic_clause* in "Analytic Functions" in *Oracle Database SQL Language Reference*.)
  - For Regression, specify `FOR` *expr*, where *expr* is an expression that identifies a target column that has a numeric data type.
  - For Classification, specify `FOR` *expr*, where *expr* is an expression that identifies a target column that has a character data type.
  - For Anomaly Detection, specify the keywords `OF ANOMALY`.

The syntax of the `PREDICTION` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

The *mining_attribute_clause* identifies the column attributes to use as predictors for scoring.

- If you specify `USING *`, then all the relevant attributes present in the input row are used.
- If you invoke the function with the analytic syntax, then the *mining_attribute_clause* is used both for building the transient models and for scoring.
- If you invoke the function with a pre-defined model, then the *mining_attribute_clause* should include all or some of the attributes that were used to create the model. The following conditions apply:
  - If the *mining_attribute_clause* includes an attribute with the same name but a different data type from the one that was used to create the model, then the data type is converted to the type expected by the model.
  - If you specify more attributes for scoring than were used to create the model, then the extra attributes are silently ignored.
  - If you specify fewer attributes for scoring than were used to create the model, then scoring is performed on a best-effort basis.

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring.
- *Oracle Machine Learning for SQL Concepts* for information about predictive Oracle Machine Learning for SQL.
- Appendix C in *Oracle Database Globalization Support Guide* for the collation derivation rules, which define the collation assigned to the return value of `PREDICTION` when it is a character value

Note:

The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

In this example, the model `dt_sh_clas_sample` predicts the gender and age of customers who are most likely to use an affinity card (target = 1). The `PREDICTION` function takes into account the cost matrix associated with the model and uses marital status, education, and household size as predictors.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
   FROM mining_data_apply_v
   WHERE PREDICTION(dt_sh_clas_sample COST MODEL
      USING cust_marital_status, education, household_size) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;

CUST_GENDER         CNT    AVG_AGE
------------ ---------- ----------
F                   170         38
M                   685         42
```

The cost matrix associated with the model `dt_sh_clas_sample` is stored in the table `dt_sh_sample_costs`. The cost matrix specifies that the misclassification of 1 is 8 times more costly than the misclassification of 0.

```
SQL> select * from dt_sh_sample_cost;

ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE         COST
------------------- ---------------------- ------------
                  0                      0    .000000000
                  0                      1   1.000000000
                  1                      0   8.000000000
                  1                      1    .000000000
```

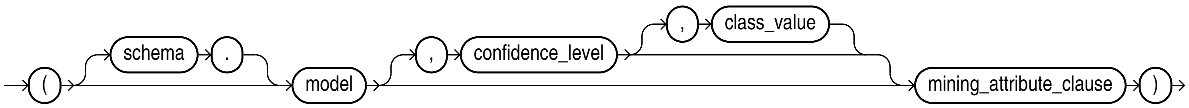
Analytic Example

In this example, dynamic regression is used to predict the age of customers who are likely to use an affinity card. The query returns the 3 customers whose predicted age is most different from the actual. The query includes information about the predictors that have the greatest influence on the prediction.

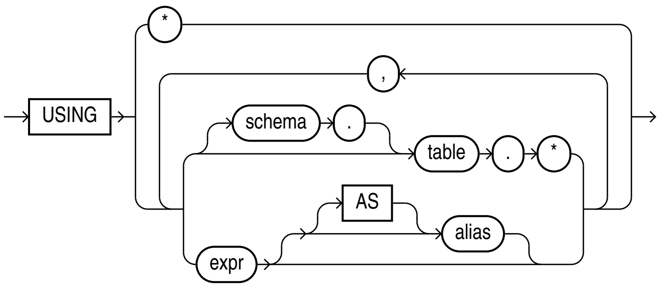
```
SELECT cust_id, age, pred_age, age-pred_age age_diff, pred_det FROM
   (SELECT cust_id, age, pred_age, pred_det,
          RANK() OVER (ORDER BY ABS(age-pred_age) desc) rnk FROM
   (SELECT cust_id, age,
           PREDICTION(FOR age USING *) OVER () pred_age,
           PREDICTION_DETAILS(FOR age ABS USING *) OVER () pred_det
    FROM mining_data_apply_v))
  WHERE rnk <= 3;

CUST_ID  AGE PRED_AGE AGE_DIFF PRED_DET
------- ---- -------- -------- --------
----------------------------------------------------------
 100910   80    40.67    39.33 <Details algorithm="Support Vector Machines">
                               <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".059"
                                rank="1"/>
                               <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                                rank="2"/>
                               <Attribute name="AFFINITY_CARD" actualValue="0" weight=".059"
                                rank="3"/>
                               <Attribute name="FLAT_PANEL_MONITOR" actualValue="1" weight=".059"
                                rank="4"/>
                               <Attribute name="YRS_RESIDENCE" actualValue="4" weight=".059"
                                rank="5"/>
                               </Details>

 101285    79   42.18    36.82 <Details algorithm="Support Vector Machines">
                               <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".059"
                                rank="1"/>
                               <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".059"
                                rank="2"/>
                               <Attribute name="CUST_MARITAL_STATUS" actualValue="Mabsent"
                                weight=".059" rank="3"/>
                               <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                                rank="4"/>
                               <Attribute name="OCCUPATION" actualValue="Prof." weight=".059"
                                rank="5"/>
                               </Details>

 100694     77  41.04    35.96 <Details algorithm="Support Vector Machines">
                               <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".059"
                                rank="1"/>
                               <Attribute name="EDUCATION" actualValue="&lt; Bach." weight=".059"
                                rank="2"/>
                               <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                                rank="3"/>
                               <Attribute name="CUST_ID" actualValue="100694" weight=".059"
                                rank="4"/>
                               <Attribute name="COUNTRY_NAME" actualValue="United States of
                                America" weight=".059" rank="5"/>
                               </Details>
```

44.13 PREDICTION_BOUNDS

Syntax



***mining_attribute_clause*::=**



Purpose

`PREDICTION_BOUNDS` applies a Generalized Linear Model (GLM) to predict a class or a value for each row in the selection. The function returns the upper and lower bounds of each prediction in a varray of objects with fields `UPPER` and `LOWER`.

GLM can perform either regression or binary classification:

- The bounds for regression refer to the predicted target value. The data type of `UPPER` and `LOWER` is the data type of the target.
- The bounds for binary classification refer to the probability of either the predicted target class or the specified *`class_value`*. The data type of `UPPER` and `LOWER` is `BINARY_DOUBLE`.

If the model was built using ridge regression, or if the covariance matrix is found to be singular during the build, then `PREDICTION_BOUNDS` returns `NULL` for both bounds.

`confidence_level` is a number in the range (0,1). The default value is 0.95. You can specify *`class_value`* while leaving *`confidence_level`* at its default by specifying `NULL` for *`confidence_level`*.

The syntax of the `PREDICTION_BOUNDS` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

`mining_attribute_clause` identifies the column attributes to use as predictors for scoring. This clause behaves as described for the `PREDICTION` function. (Note that the reference to analytic syntax does not apply.) See "mining_attribute_clause".

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring
- *Oracle Machine Learning for SQL Concepts* for information about Generalized Linear Models

Note:

The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

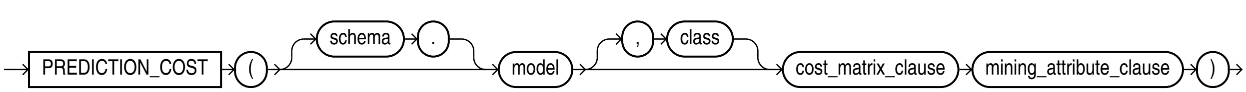
The following example returns the distribution of customers whose ages are predicted with 98% confidence to be greater than 24 and less than 46.

```
SELECT count(cust_id) cust_count, cust_marital_status
  FROM (SELECT cust_id, cust_marital_status
    FROM mining_data_apply_v
    WHERE PREDICTION_BOUNDS(glmr_sh_regr_sample,0.98 USING *).LOWER > 24 AND
          PREDICTION_BOUNDS(glmr_sh_regr_sample,0.98 USING *).UPPER < 46)
    GROUP BY cust_marital_status;

    CUST_COUNT CUST_MARITAL_STATUS
-------------- --------------------
            46 NeverM
             7 Mabsent
             5 Separ.
            35 Divorc.
            72 Married
```

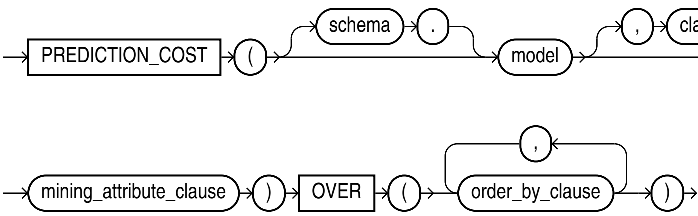
44.14 PREDICTION_COST

Syntax

***prediction_cost*::=**

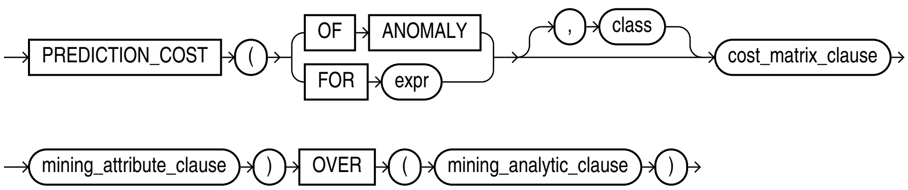


***prediction_cost_ordered*::=**

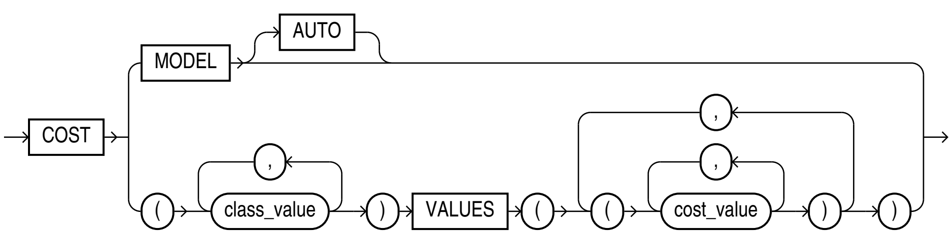


Analytic Syntax

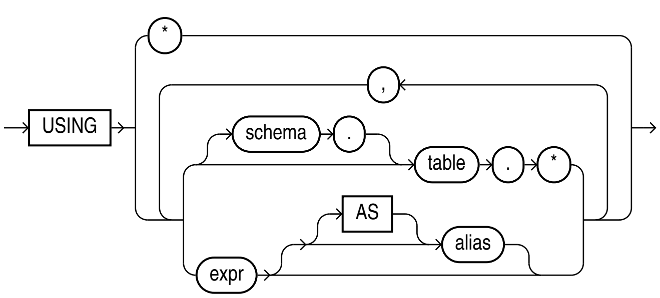
***prediction_cost_analytic*::=**



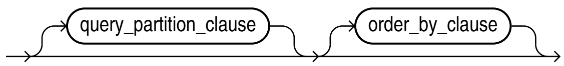
***cost_matrix_clause*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



See Also:

"Analytic Functions" for information on the syntax, semantics, and restrictions of the `mining_analytic_clause`

Purpose

`PREDICTION_COST` returns a cost for each row in the selection. The cost refers to the lowest cost class or to the specified *`class`*. The cost is returned as a `BINARY_DOUBLE`.

`PREDICTION_COST` can perform classification or anomaly detection. For classification, the returned cost refers to a predicted target class. For anomaly detection, the returned cost refers to a classification of `1` (for typical rows) or `0` (for anomalous rows).

You can use `PREDICTION_COST` in conjunction with the `PREDICTION` function to obtain the prediction and the cost of the prediction.

cost_matrix_clause

Costs are a biasing factor for minimizing the most harmful kinds of misclassifications. For example, false positives might be considered more costly than false negatives. Costs are specified in a cost matrix that can be associated with the model or defined inline in a `VALUES` clause. All classification algorithms can use costs to influence scoring.

Decision Tree is the only algorithm that can use costs to influence the model build. The cost matrix used to build a Decision Tree model is also the default scoring cost matrix for the model.

The following cost matrix table specifies that the misclassification of `1` is five times more costly than the misclassification of `0`.

```
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1          1
                  1                      0          5
                  1                      1          0
```

In *`cost_matrix_clause`*:

- `COST MODEL` indicates that scoring should be performed by taking into account the scoring cost matrix associated with the model. If the cost matrix does not exist, then the function returns an error.
- `COST MODEL AUTO` indicates that the existence of a cost matrix is unknown. If a cost matrix exists, then the function uses it to return the lowest cost prediction. Otherwise the function returns the highest probability prediction.

- The VALUES clause specifies an inline cost matrix for *class_value*. For example, you could specify that the misclassification of 1 is five times more costly than the misclassification of 0 as follows:

  ```
  PREDICTION (nb_model COST (0,1) VALUES ((0, 1),(1, 5)) USING *)
  ```

  If a model that has a scoring cost matrix is invoked with an inline cost matrix, then the inline costs are used.

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for more information about cost-sensitive prediction.

Syntax Choice

PREDICTION_COST can score data by applying a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax**: Use the *prediction_cost* syntax to score the data with a pre-defined model. Supply the name of a model that performs classification or anomaly detection.

  Use the *prediction_cost_ordered* syntax for a model that requires ordered data, such as an MSET-SPRT model. The *prediction_cost_ordered* syntax requires an *order_by_clause* clause.

  Restrictions on the *prediction_cost_ordered* syntax are that you cannot use it in the WHERE clause of a query. Also, you cannot use a *query_partition_clause* or a *windowing_clause* with the *prediction_cost_ordered* syntax.

  For details about the *order_by_clause*, see "Analytic Functions" in *Oracle Database SQL Language Reference*.
- **Analytic Syntax**: Use the *prediction_cost_analytic* syntax to score the data without a pre-defined model. The analytic syntax uses the *mining_analytic_clause*, which specifies whether the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See the *analytic_clause* in "Analytic Functions" in *Oracle Database SQL Language Reference*.)
  - For classification, specify FOR *expr*, where *expr* is an expression that identifies a target column that has a character data type.
  - For anomaly detection, specify the keywords OF ANOMALY.

The syntax of the PREDICTION_COST function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

The *mining_attribute_clause* identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User’s Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about classification with costs

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User’s Guide*.

Example

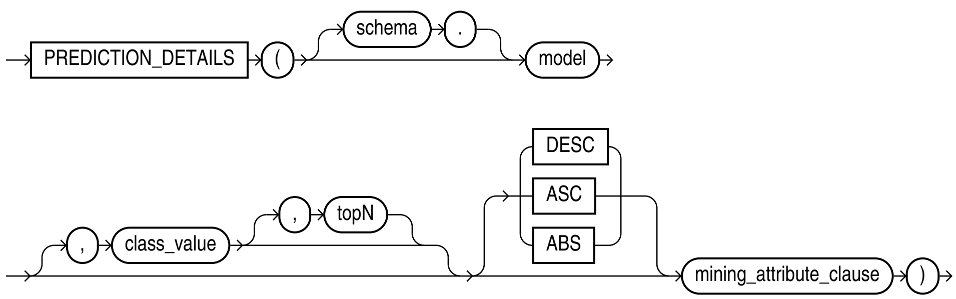
This example predicts the ten customers in Italy who would respond to the least expensive sales campaign (offering an affinity card).

```
SELECT cust_id
FROM (SELECT cust_id,rank()
       OVER (ORDER BY PREDICTION_COST(DT_SH_Clas_sample, 1 COST MODEL USING *)
            ASC, cust_id) rnk
        FROM mining_data_apply_v
        WHERE country_name = 'Italy')
  WHERE rnk <= 10
  ORDER BY rnk;

   CUST_ID
----------
    100081
    100179
    100185
    100324
    100344
    100554
    100662
    100733
    101250
    101306
```

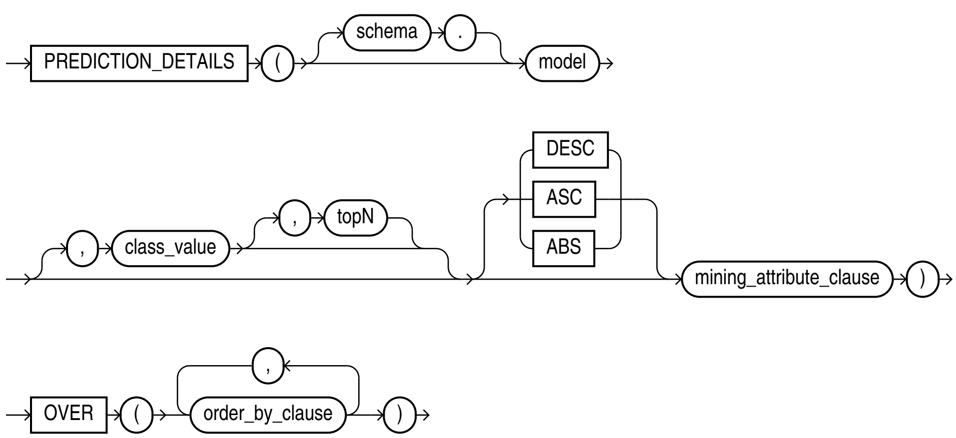
44.15 PREDICTION_DETAILS

Syntax

***prediction_details*::=**

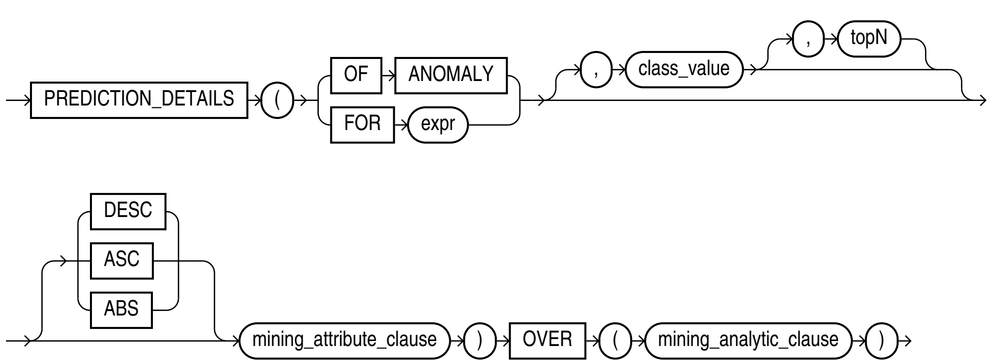


***prediction_details_ordered*::=**

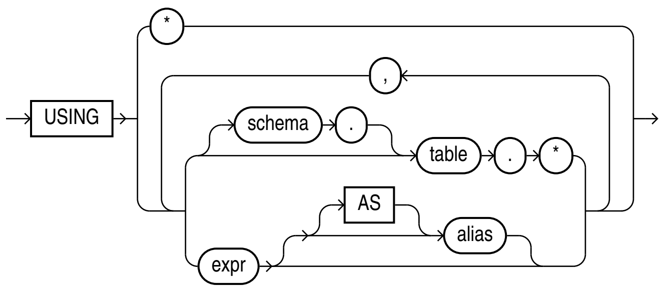


Analytic Syntax

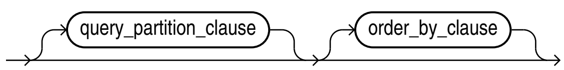
***prediction_details_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *`mining_analytic_clause`*

Purpose

`PREDICTION_DETAILS` returns prediction details for each row in the selection. The return value is an XML string that describes the attributes of the prediction.

For regression, the returned details refer to the predicted target value. For classification and anomaly detection, the returned details refer to the highest probability class or the specified *`class_value`*.

topN

If you specify a value for *`topN`*, the function returns the *`N`* attributes that have the most influence on the prediction (the score). If you do not specify *`topN`*, the function returns the 5 most influential attributes.

DESC, ASC, or ABS

The returned attributes are ordered by weight. The weight of an attribute expresses its positive or negative impact on the prediction. For regression, a positive weight indicates a higher value prediction; a negative weight indicates a lower value prediction. For classification and anomaly detection, a positive weight indicates a higher probability prediction; a negative weight indicates a lower probability prediction.

By default, `PREDICTION_DETAILS` returns the attributes with the highest positive weight (`DESC`). If you specify `ASC`, the attributes with the highest negative weight are returned. If you specify `ABS`, the attributes with the greatest weight, whether negative or positive, are returned. The results are ordered by absolute value from highest to lowest. Attributes with a zero weight are not included in the output.

Syntax Choice

`PREDICTION_DETAILS` can score the data by applying a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax**: Use the *`prediction_details`* syntax to score the data with a pre-defined model. Supply the name of a model that performs classification, regression, or anomaly detection.

  Use the *`prediction_details_ordered`* syntax for a model that requires ordered data, such as an MSET-SPRT model. The *`prediction_details_ordered`* syntax requires an *`order_by_clause`* clause.

  Restrictions on the *`prediction_details_ordered`* syntax are that you cannot use it in the `WHERE` clause of a query. Also, you cannot use a *`query_partition_clause`* or a *`windowing_clause`* with the *`prediction_details_ordered`* syntax.

  > **Note:**
  >
  > When random projections are engaged for an MSET-SPRT model., only the overall `PREDICTION` and `PREDICTION_PROBABILITY` are computed and `PREDICTION_DETAILS` are not reported.

  For details about the *`order_by_clause`*, see "Analytic Functions" in *Oracle Database SQL Language Reference*.

- **Analytic Syntax**: Use the *`prediction_details_analytic`* syntax to score the data without a pre-defined model. The analytic syntax uses *`mining_analytic_clause`*, which specifies if the data should be partitioned for multiple model builds. The *`mining_analytic_clause`* supports a *`query_partition_clause`* and an *`order_by_clause`*. (See "analytic_clause::=".)
  - For classification, specify `FOR` *`expr`*, where *`expr`* is an expression that identifies a target column that has a character data type.
  - For regression, specify `FOR` *`expr`*, where *`expr`* is an expression that identifies a target column that has a numeric data type.
  - For anomaly detection, specify the keywords `OF ANOMALY`.

The syntax of the `PREDICTION_DETAILS` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

`mining_attribute_clause` identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

See Also:

- *Oracle Machine Learning for SQL User's Guide* for information about scoring.
- *Oracle Machine Learning for SQL Concepts* for information about predictive Oracle Machine Learning for SQL.

Note:

The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

This example uses the model `svmr_sh_regr_sample` to score the data. The query returns the three attributes that have the greatest influence on predicting a higher value for customer age.

```
SELECT PREDICTION_DETAILS(svmr_sh_regr_sample, null, 3 USING *) prediction_details
    FROM mining_data_apply_v
    WHERE cust_id = 100001;

PREDICTION_DETAILS
---------------------------------------------------------------------------------------
<Details algorithm="Support Vector Machines">
<Attribute name="CUST_MARITAL_STATUS" actualValue="Widowed" weight=".361" rank="1"/>
<Attribute name="CUST_GENDER" actualValue="F" weight=".14" rank="2"/>
<Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".135" rank="3"/>
</Details>
```

Analytic Syntax

This example dynamically identifies customers whose age is not typical for the data. The query returns the attributes that predict or detract from a typical age.

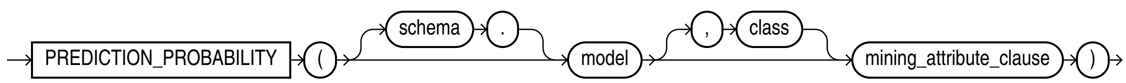
```
SELECT cust_id, age, pred_age, age-pred_age age_diff, pred_det
    FROM (SELECT cust_id, age, pred_age, pred_det,
          RANK() OVER (ORDER BY ABS(age-pred_age) DESC) rnk
          FROM (SELECT cust_id, age,
             PREDICTION(FOR age USING *) OVER () pred_age,
             PREDICTION_DETAILS(FOR age ABS USING *) OVER () pred_det
             FROM mining_data_apply_v))
    WHERE rnk <= 5;

CUST_ID AGE PRED_AGE AGE_DIFF  PRED_DET
------- --- -------- -------- ------------------------------------------------------------------
 100910  80    40.67    39.33 <Details algorithm="Support Vector Machines">
                              <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".059"
                               rank="1"/>
                              <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                               rank="2"/>
```

```
                          <Attribute name="AFFINITY_CARD" actualValue="0" weight=".059"
                           rank="3"/>
                          <Attribute name="FLAT_PANEL_MONITOR" actualValue="1" weight=".059"
                           rank="4"/>
                          <Attribute name="YRS_RESIDENCE" actualValue="4" weight=".059"
                           rank="5"/>
                          </Details>

101285  79    42.18    36.82  <Details algorithm="Support Vector Machines">
                              <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".059"
                               rank="1"/>
                              <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".059"
                               rank="2"/>
                              <Attribute name="CUST_MARITAL_STATUS" actualValue="Mabsent"
                               weight=".059" rank="3"/>
                              <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                               rank="4"/>
                              <Attribute name="OCCUPATION" actualValue="Prof." weight=".059"
                               rank="5"/>
                              </Details>

100694   77    41.04    35.96  <Details algorithm="Support Vector Machines">
                               <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
                                weight=".059" rank="1"/>
                               <Attribute name="EDUCATION" actualValue="&lt; Bach." weight=".059"
                                rank="2"/>
                               <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                                rank="3"/>
                               <Attribute name="CUST_ID" actualValue="100694" weight=".059"
                                rank="4"/>
                               <Attribute name="COUNTRY_NAME" actualValue="United States of
                                America" weight=".059" rank="5"/>
                               </Details>

100308  81    45.33    35.67  <Details algorithm="Support Vector Machines">
                              <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".059"
                               rank="1"/>
                              <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                               rank="2"/>
                              <Attribute name="HOUSEHOLD_SIZE" actualValue="2" weight=".059"
                               rank="3"/>
                              <Attribute name="FLAT_PANEL_MONITOR" actualValue="1" weight=".059"
                               rank="4"/>
                              <Attribute name="CUST_GENDER" actualValue="F" weight=".059"
                               rank="5"/>
                              </Details>

101256  90    54.39    35.61  <Details algorithm="Support Vector Machines">
                              <Attribute name="YRS_RESIDENCE" actualValue="9" weight=".059"
                               rank="1"/>
                              <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".059"
                               rank="2"/>
                              <Attribute name="EDUCATION" actualValue="&lt; Bach." weight=".059"
                               rank="3"/>
                              <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".059"
                               rank="4"/>
                              <Attribute name="COUNTRY_NAME" actualValue="United States of
                               America" weight=".059" rank="5"/>
                              </Details>
```

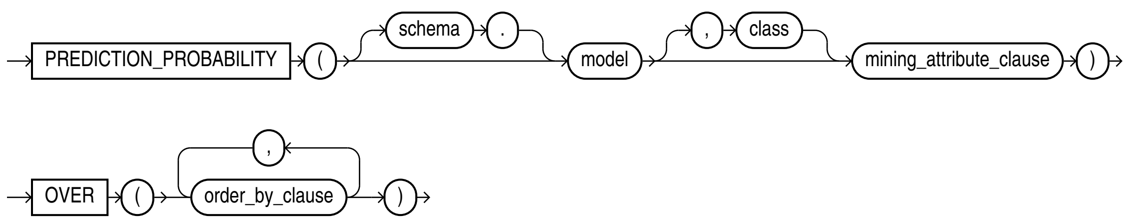
44.16 PREDICTION_PROBABILITY

Syntax

***prediction_probability*::=**

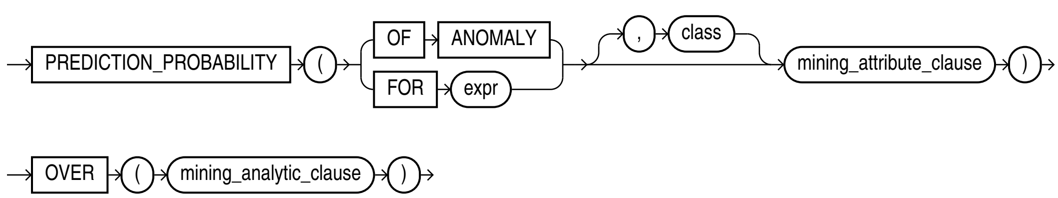


***prediction_probability_ordered*::=**

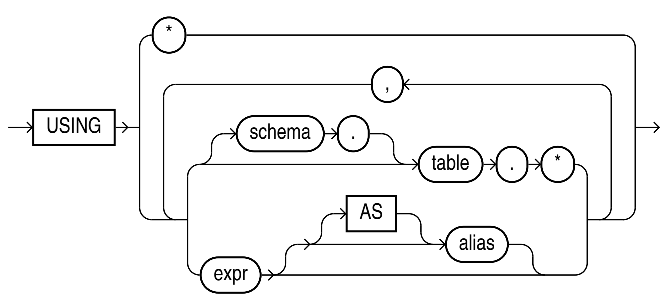


Analytic Syntax

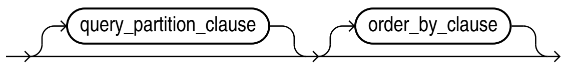
***prediction_prob_analytic*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::=**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *mining_analytic_clause*

Purpose

PREDICTION_PROBABILITY returns a probability for each row in the selection. The probability refers to the highest probability class or to the specified *class*. The data type of the returned probability is BINARY_DOUBLE.

PREDICTION_PROBABILITY can perform classification or anomaly detection. For classification, the returned probability refers to a predicted target class. For anomaly detection, the returned probability refers to a classification of 1 (for typical rows) or 0 (for anomalous rows).

You can use PREDICTION_PROBABILITY in conjunction with the PREDICTION function to obtain the prediction and the probability of the prediction.

Syntax Choice

PREDICTION_PROBABILITY can score the data by applying a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax**: Use the *prediction_probability* syntax to score the data with a pre-defined model. Supply the name of a model that performs classification or anomaly detection.

  Use the *prediction_probability_ordered* syntax for a model that requires ordered data, such as an MSET-SPRT model. The *prediction_probability_ordered* syntax requires an *order_by_clause* clause.

  Restrictions on the *prediction_probability_ordered* syntax are that you cannot use it in the WHERE clause of a query. Also, you cannot use a *query_partition_clause* or a *windowing_clause* with the *prediction_probability_ordered* syntax.

  For details about the *order_by_clause*, see "Analytic Functions" in *Oracle Database SQL Language Reference*.

- **Analytic Syntax**: Use the analytic syntax to score the data without a pre-defined model. The analytic syntax uses *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)

  – For classification, specify FOR *expr*, where *expr* is an expression that identifies a target column that has a character data type.

  – For anomaly detection, specify the keywords OF ANOMALY.

The syntax of the `PREDICTION_PROBABILITY` function can use an optional `GROUPING` hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

`mining_attribute_clause` identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *`mining_attribute_clause`* behaves as described for the `PREDICTION` function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about predictive Oracle Machine Learning for SQL.

> **Note:**
>
> The following examples are excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

The following example returns the 10 customers living in Italy who are most likely to use an affinity card.

```
SELECT cust_id FROM (
   SELECT cust_id
   FROM mining_data_apply_v
   WHERE country_name = 'Italy'
   ORDER BY PREDICTION_PROBABILITY(DT_SH_Clas_sample, 1 USING *)
      DESC, cust_id)
   WHERE rownum < 11;

   CUST_ID
----------
    100081
    100179
    100185
    100324
    100344
    100554
    100662
    100733
    101250
    101306
```

Analytic Example

This example identifies rows that are most atypical in the data in `mining_data_one_class_v`. Each type of marital status is considered separately so that the most anomalous rows per marital status group are returned.

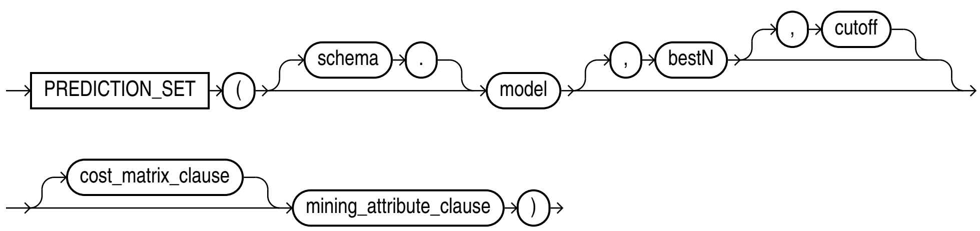
The query returns three attributes that have the most influence on the determination of anomalous rows. The `PARTITION BY` clause causes separate models to be built and applied for each marital status. Because there is only one record with status Mabsent, no model is created for that partition (and no details are provided).

```
SELECT cust_id, cust_marital_status, rank_anom, anom_det FROM
    (SELECT cust_id, cust_marital_status, anom_det,
            rank() OVER (PARTITION BY CUST_MARITAL_STATUS
                         ORDER BY ANOM_PROB DESC,cust_id) rank_anom FROM
     (SELECT cust_id, cust_marital_status,
            PREDICTION_PROBABILITY(OF ANOMALY, 0 USING *)
              OVER (PARTITION BY CUST_MARITAL_STATUS) anom_prob,
            PREDICTION_DETAILS(OF ANOMALY, 0, 3 USING *)
              OVER (PARTITION BY CUST_MARITAL_STATUS) anom_det
     FROM mining_data_one_class_v
    ))
   WHERE rank_anom < 3 order by 2, 3;

CUST_ID CUST_MARITAL_STATUS RANK_ANOM  ANOM_DET
------- ------------------- ---------- -----------------------------------------------------------
102366  Divorc.             1          <Details algorithm="Support Vector Machines" class="0">
                                       <Attribute name="COUNTRY_NAME" actualValue="United Kingdom"
                                        weight=".069" rank="1"/>
                                       <Attribute name="AGE" actualValue="28" weight=".013"
                                        rank="2"/>
                                       <Attribute name="YRS_RESIDENCE" actualValue="4"
                                        weight=".006" rank="3"/>
                                       </Details>

101817  Divorc.             2          <Details algorithm="Support Vector Machines" class="0">
                                       <Attribute name="YRS_RESIDENCE" actualValue="8"
                                        weight=".018" rank="1"/>
                                       <Attribute name="EDUCATION" actualValue="PhD" weight=".007"
                                        rank="2"/>
                                       <Attribute name="CUST_INCOME_LEVEL" actualValue="K:
                                        250\,000 - 299\,999" weight=".006" rank="3"/>
                                       </Details>

101713  Mabsent             1

101790  Married             1          <Details algorithm="Support Vector Machines" class="0">
                                       <Attribute name="COUNTRY_NAME" actualValue="Canada"
                                        weight=".063" rank="1"/>
                                       <Attribute name="EDUCATION" actualValue="7th-8th"
                                        weight=".011" rank="2"/>
                                       <Attribute name="HOUSEHOLD_SIZE" actualValue="4-5"
                                        weight=".011" rank="3"/>
                                       </Details>
. . .
```

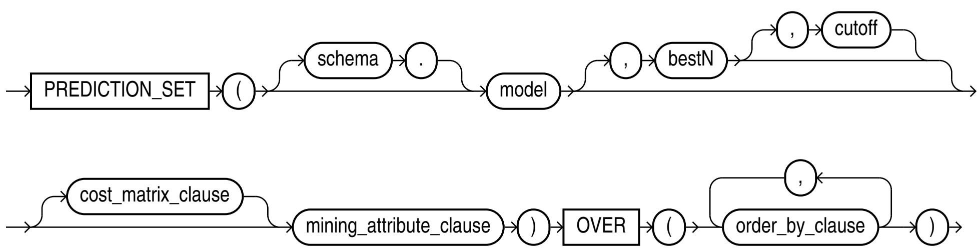
44.17 PREDICTION_SET

Syntax

***prediction_set*::=**

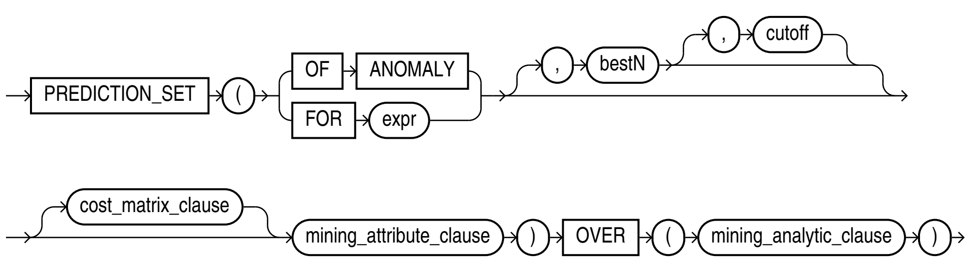


***prediction_set_ordered*::=**

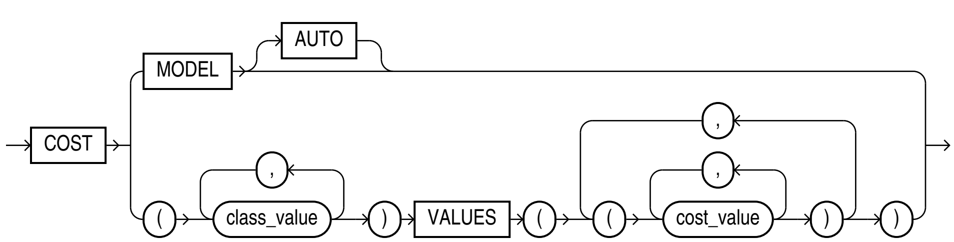


Analytic Syntax

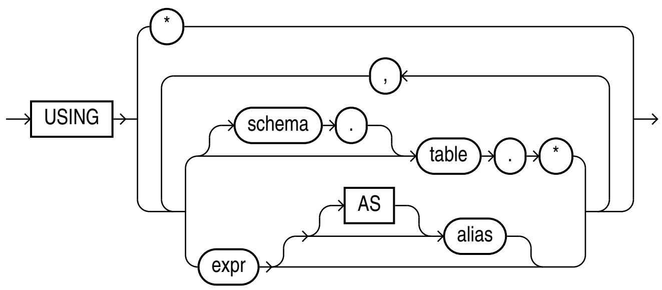
***prediction_set_analytic*::=**



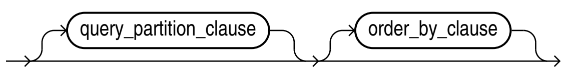
***cost_matrix_clause*::=**



***mining_attribute_clause*::=**



***mining_analytic_clause*::-**



> **See Also:**
>
> "Analytic Functions" for information on the syntax, semantics, and restrictions of *`mining_analytic_clause`*

Purpose

`PREDICTION_SET` returns a set of predictions with either probabilities or costs for each row in the selection. The return value is a varray of objects with field names `PREDICTION_ID` and `PROBABILITY` or `COST`. The data type of the `PREDICTION` field depends on the target value type used during the build of the model; the probability and cost fields are `BINARY_DOUBLE`.

`PREDICTION_SET` can perform classification or anomaly detection. For classification, the return value refers to a predicted target class. For anomaly detection, the return value refers to a classification of `1` (for typical rows) or `0` (for anomalous rows).

bestN and cutoff

You can specify *`bestN`* and *`cutoff`* to limit the number of predictions returned by the function. By default, both *`bestN`* and *`cutoff`* are null and all predictions are returned.

- *`bestN`* is the *`N`* predictions that are either the most probable or the least costly. If multiple predictions share the *`N`*th probability or cost, then the function chooses one of them.
- *`cutoff`* is a value threshold. Only predictions with probability greater than or equal to *`cutoff`*, or with cost less than or equal to *`cutoff`*, are returned. To filter by *`cutoff`* only, specify `NULL` for *`bestN`*. If the function uses a *`cost_matrix_clause`* with `COST MODEL AUTO`, then *`cutoff`* is ignored.

You can specify *`bestN`* with *`cutoff`* to return up to the *`N`* most probable predictions that are greater than or equal to *`cutoff`*. If costs are used, specify *`bestN`* with *`cutoff`* to return up to the *`N`* least costly predictions that are less than or equal to cutoff.

cost_matrix_clause

You can specify *cost_matrix_clause* as a biasing factor for minimizing the most harmful kinds of misclassifications. *cost_matrix_clause* behaves as described for "PREDICTION_COST ".

Syntax Choice

PREDICTION_SET can score the data by applying a mining model object to the data, or it can dynamically mine the data by executing an analytic clause that builds and applies one or more transient mining models. Choose **Syntax** or **Analytic Syntax**:

- **Syntax**: Use the *prediction_set* syntax to score the data with a pre-defined model. Supply the name of a model that performs classification or anomaly detection.

  Use the *prediction_set_ordered* syntax for a model that requires ordered data, such as an MSET-SPRT model. The *prediction_set_ordered* syntax requires an *order_by_clause* clause.

  Restrictions on the *prediction_set_ordered* syntax are that you cannot use it in the WHERE clause of a query. Also, you cannot use a *query_partition_clause* or a *windowing_clause* with the *prediction_set_ordered* syntax.

  For details about the *order_by_clause*, see "Analytic Functions" in *Oracle Database SQL Language Reference*.

- **Analytic Syntax**: Use the analytic syntax to score the data without a pre-defined model. The analytic syntax uses *mining_analytic_clause*, which specifies if the data should be partitioned for multiple model builds. The *mining_analytic_clause* supports a *query_partition_clause* and an *order_by_clause*. (See "analytic_clause::=".)
  - For classification, specify FOR *expr*, where *expr* is an expression that identifies a target column that has a character data type.
  - For anomaly detection, specify the keywords OF ANOMALY.

The syntax of the PREDICTION_SET function can use an optional GROUPING hint when scoring a partitioned model. See GROUPING Hint.

mining_attribute_clause

mining_attribute_clause identifies the column attributes to use as predictors for scoring. When the function is invoked with the analytic syntax, these predictors are also used for building the transient models. The *mining_attribute_clause* behaves as described for the PREDICTION function. (See "mining_attribute_clause".)

> **See Also:**
>
> - *Oracle Machine Learning for SQL User's Guide* for information about scoring.
> - *Oracle Machine Learning for SQL Concepts* for information about predictive Oracle Machine Learning for SQL.

> **Note:**
>
> The following example is excerpted from the Oracle Machine Learning for SQL examples. For more information about the examples, see Appendix A in *Oracle Machine Learning for SQL User's Guide*.

Example

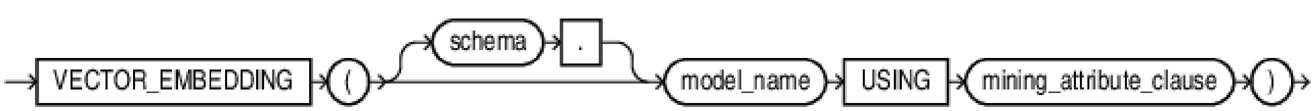
This example lists the probability and cost that customers with ID less than 100006 will use an affinity card. This example has a binary target, but such a query is also useful for multiclass classification such as low, medium, and high.

```
SELECT T.cust_id, S.prediction, S.probability, S.cost
  FROM (SELECT cust_id,
               PREDICTION_SET(dt_sh_clas_sample COST MODEL USING *) pset
          FROM mining_data_apply_v
         WHERE cust_id < 100006) T,
       TABLE(T.pset) S
ORDER BY cust_id, S.prediction;

   CUST_ID PREDICTION  PROBABILITY         COST
---------- ---------- ------------ ------------
    100001          0   .966183575   .270531401
    100001          1   .033816425   .966183575
    100002          0   .740384615  2.076923077
    100002          1   .259615385   .740384615
    100003          0   .909090909   .727272727
    100003          1   .090909091   .909090909
    100004          0   .909090909   .727272727
    100004          1   .090909091   .909090909
    100005          0   .272357724  5.821138211
    100005          1   .727642276   .272357724
```

44.18 VECTOR_EMBEDDING

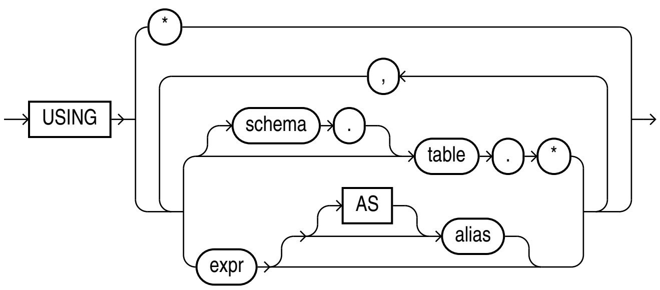
Syntax

***vector_embedding*::=**



Analytic Syntax

***mining_attribute_clause*::=**



Purpose

Use `VECTOR_EMBEDDING` if you want to generate a single vector embedding for different data types. To get embedding, this function uses pretrained ONNX embedding machine learning models.

Syntax Choice

The function accepts the following types as input:

`VARCHAR2` for text embedding models. Oracle automatically converts any other type to `VARCHAR2` except for `NCLOB`, which will be automatically converted to `NVARCHAR2`. Oracle does not expect values whose textual representation exceeds the maximum size of a `VARCHAR2`, since embedding models support only text that translates to a couple thousand tokens. An attribute with a type that has no conversion to `VARCHAR2` results in a SQL compilation error.

The function always returns a `VECTOR` type, whose dimension is dictated by the model itself. The model stores the dimension information in metadata within the data dictionary.

You can use `VECTOR_EMBEDDING` in `SELECT` clauses, in predicates, and as an operand for SQL operations accepting a `VECTOR` type.

> **See Also:**
>
> Vector Functions

Parameters:

`model_name` refers to the name of the imported embedding model that implements the embedding machine learning function.

Use the first syntax to score the data with a pre-defined model. Supply the name of an embedding model.

mining_attribute_clause

The `mining_attribute_clause` is one of the following:

- The *`mining_attribute_clause`* argument identifies the column attributes to use as predictors for scoring. This is used as a convenience, as embedding operator only accepts single input value.
- `USING *` : all the relevant attributes present in the input (supplied in JSON metadata) are used. This is used as a convenience. For an embedding model, the operator only takes one input value as embedding models have only one column.
- `USING <column expression> [AS <alias>] [, <column expression> [AS <alias>] ]` : all the relevant attributes present in the comma-separated list of column expressions are used. This syntax is consistent with the syntax of other machine learning operators. You may specify more than one attribute, however, the embedding model only takes one relevant input. Therefore, you must specify a single mining attribute.

The operator always returns a `VECTOR` type, whose dimension is dictated by the model itself. The model stores the dimension information in metadata within the data dictionary.

The operator accepts values of the following types for its input attribute:

Example

The following example generates vector embeddings with "hello" as the input, utilizing the pretrained ONNX format model `my_embedding_model.onnx` imported into the Database. For complete example, see Import ONNX Models and Generate Embeddings.

```
SELECT TO_VECTOR(VECTOR_EMBEDDING(model USING 'hello' as data)) AS embedding;
-----------------------------------------------------------------------------
---
[-9.76553112E-002,-9.89954844E-002,7.69771636E-003,-4.16760892E-003,-9.693056
34E-002,
-3.01141385E-002,-2.63396613E-002,-2.98553891E-002,5.96499592E-002,4.13885899
E-002,
5.32859489E-002,6.57707453E-002,-1.47056757E-002,-4.18472625E-002,4.1588001E-
002,
-2.86354572E-002,-7.56499246E-002,-4.16395674E-003,-1.52879998E-001,6.6001057
6E-002,
-3.9013084E-002,3.15719917E-002,1.2428958E-002,-2.47651711E-002,-1.16851285E-
001,
-7.82847106E-002,3.34323719E-002,8.03267583E-002,1.70483496E-002,-5.42407483E
-002,
6.54291287E-002,-4.81935125E-003,6.11041225E-002,6.64106477E-003,-5.47
```