Contents

Preface

Audience vii
Documentation Accessibility vii
Related Resources vii
Conventions viii

1 About Oracle Machine Learning for Python

What Is Oracle Machine Learning for Python? 1-1
Advantages of Oracle Machine Learning for Python 1-2
Transparently Convert Python to SQL 1-3
Manage Resources Used by OML4Py 1-5
About the Python Components and Libraries in OML4Py 1-8

2 Install OML4Py for On-Premises Databases

OML4Py On Premises System Requirements 2-1
Build and Install Python for Linux for On-Premises Databases 2-2
Install the Required Supporting Packages for Linux for On-Premises Databases 2-3
Install OML4Py Server for On-Premises Oracle Database 2-4
  Install OML4Py Server for Linux for On-Premises Oracle Database 21c 2-4
  Verify OML4Py Server Installation for On-Premises Database 2-5
  Grant Users the Required Privileges for On-Premises Database 2-6
  Create New Users for On-Premises Oracle Database 2-7
  Uninstall the OML4Py Server from an On-Premises Database 2-8
Install OML4Py Client for On-Premises Databases 2-9
  Install Oracle Instant Client and the OML4Py Client for Linux 2-9
    Install Oracle Instant Client for Linux for On-Premises Databases 2-9
    Install OML4Py Client for Linux for On-Premises Databases 2-10
  Verify OML4Py Client Installation for On-Premises Databases 2-13
  Uninstall the OML4Py Client for On-Premises Databases 2-13
### 3 Get Started with Oracle Machine Learning for Python

- Use OML4Py with Oracle Autonomous Database 3-1
- Use OML4Py with an On-Premises Oracle Database 3-1
  - About Connecting to an On-Premises Oracle Database 3-1
  - About Oracle Wallets 3-3
  - Connect to an Oracle Database 3-4
- Move Data Between the Database and a Python Session 3-8
  - About Moving Data Between the Database and a Python Session 3-8
  - Push Local Python Data to the Database 3-9
  - Pull Data from the Database to a Local Python Session 3-11
  - Create a Python Proxy Object for a Database Object 3-13
  - Create a Persistent Database Table from a Python Data Set 3-15
- Save Python Objects in the Database 3-19
  - About OML4Py Datastores 3-20
  - Save Objects to a Datastore 3-21
  - Load Saved Objects From a Datastore 3-24
  - Get Information About Datastores 3-25
  - Get Information About Datastore Objects 3-27
  - Delete Datastore Objects 3-28
  - Manage Access to Stored Objects 3-30

### 4 Prepare and Explore Data

- Prepare Data 4-1
  - About Preparing Data in the Database 4-1
  - Select Data 4-3
  - Combine Data 4-8
  - Clean Data 4-13
  - Split Data 4-15
- Explore Data 4-17
  - About the Exploratory Data Analysis Methods 4-17
  - Correlate Data 4-19
  - Cross-Tabulate Data 4-21
  - Mutate Data 4-23
  - Sort Data 4-26
  - Summarize Data 4-29
  - Render Graphics 4-31
## 5 OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Machine Learning Classes and Algorithms</td>
<td>5-1</td>
</tr>
<tr>
<td>About Model Settings</td>
<td>5-3</td>
</tr>
<tr>
<td>Shared Settings</td>
<td>5-4</td>
</tr>
<tr>
<td>Export Oracle Machine Learning for Python Models</td>
<td>5-6</td>
</tr>
<tr>
<td>Model Explainability</td>
<td>5-10</td>
</tr>
<tr>
<td>Attribute Importance</td>
<td>5-16</td>
</tr>
<tr>
<td>Association Rules</td>
<td>5-19</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>5-25</td>
</tr>
<tr>
<td>Expectation Maximization</td>
<td>5-32</td>
</tr>
<tr>
<td>Explicit Semantic Analysis</td>
<td>5-47</td>
</tr>
<tr>
<td>Generalized Linear Model</td>
<td>5-53</td>
</tr>
<tr>
<td>k-Means</td>
<td>5-63</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>5-69</td>
</tr>
<tr>
<td>Neural Network</td>
<td>5-77</td>
</tr>
<tr>
<td>Random Forest</td>
<td>5-87</td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td>5-95</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>5-101</td>
</tr>
</tbody>
</table>

## 6 Automated Machine Learning

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Automated Machine Learning</td>
<td>6-1</td>
</tr>
<tr>
<td>Algorithm Selection</td>
<td>6-6</td>
</tr>
<tr>
<td>Feature Selection</td>
<td>6-8</td>
</tr>
<tr>
<td>Model Tuning</td>
<td>6-11</td>
</tr>
<tr>
<td>Model Selection</td>
<td>6-16</td>
</tr>
</tbody>
</table>

## 7 Embedded Python Execution

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Embedded Python Execution and the Script Repository</td>
<td>7-1</td>
</tr>
<tr>
<td>Run a User-Defined Python Function</td>
<td>7-3</td>
</tr>
<tr>
<td>Run a User-Defined Python Function on the Specified Data</td>
<td>7-5</td>
</tr>
<tr>
<td>Run a Python Function on Data Grouped By Column Values</td>
<td>7-8</td>
</tr>
<tr>
<td>Run a User-Defined Python Function on Sets of Rows</td>
<td>7-12</td>
</tr>
<tr>
<td>Run a User-Defined Python Function Multiple Times</td>
<td>7-16</td>
</tr>
<tr>
<td>Save and Manage User-Defined Python Functions in the Script Repository</td>
<td>7-17</td>
</tr>
<tr>
<td>About the Script Repository</td>
<td>7-18</td>
</tr>
<tr>
<td>Create and Store a User-Defined Python Function</td>
<td>7-18</td>
</tr>
<tr>
<td>List Available User-Defined Python Functions</td>
<td>7-23</td>
</tr>
<tr>
<td>Load a User-Defined Python Function</td>
<td>7-24</td>
</tr>
</tbody>
</table>
8  SQL for Embedded Python Execution

About the SQL Interface for Embedded Python Execution 8-1
pyqEval Function 8-2
pyqTableEval Function 8-5
pyqRowEval Function 8-8
pyqGroupEval Function 8-12
sys.pyqScriptCreate Procedure 8-15
sys.pyqScriptDrop Procedure 8-17
ALL_PYQ_DATASTORE_CONTENTS View 8-18
ALL_PYQ_DATASTORES View 8-19
ALL_PYQ_SCRIPTS View 8-21
USER_PYQ_DATASTORES View 8-22
USER_PYQ_SCRIPTS View 8-23

Index
Preface

This publication describes Oracle Machine Learning for Python (OML4Py) and how to use it.

Audience

This document is intended for those who want to run Python commands for statistical, machine learning, and graphical analysis on data stored in or accessible through Oracle Autonomous Database using a Python API. Use of Oracle Machine Learning for Python requires knowledge of Python and of Oracle Autonomous Database.

This document is intended for those who want to run Python commands for statistical, machine learning, and graphical analysis on data stored in or accessible through Oracle Autonomous Database or Oracle Database on premises using a Python API. Use of Oracle Machine Learning for Python requires knowledge of Python and of Oracle Autonomous Database or Oracle Database on premises.

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.

Related Resources

Related documentation is in the following publications:

- Oracle Machine Learning for Python API Reference
- Oracle Machine Learning for Python Known Issues
- Oracle Machine Learning for Python Licensing Information User Manual
- REST API for Embedded Python Execution
- Get Started with Notebooks for Data Analysis and Data Visualization in Using Oracle Machine Learning Notebooks
- Oracle Machine Learning AutoML User Interface
- REST API for Oracle Machine Learning Services
For more information, see these Oracle resources:

- Oracle Machine Learning Technologies
- Oracle Autonomous Database

## Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
About Oracle Machine Learning for Python

The following topics describe Oracle Machine Learning for Python (OML4Py) and its advantages for the Python user.

- What Is Oracle Machine Learning for Python?
- Advantages of Oracle Machine Learning for Python
- Transparently Convert Python to SQL
- Manage Resources Used by OML4Py
- About the Python Components and Libraries in OML4Py

What Is Oracle Machine Learning for Python?

Oracle Machine Learning for Python (OML4Py) enables you to run Python commands for data transformations and for statistical, machine learning, and graphical analysis on data stored in or accessible through an Oracle database using a Python API.

OML4Py is a Python module that enables Python users to manipulate data in database tables and views using Python syntax. OML4Py functions and methods transparently translate a select set of Python functions into SQL for in-database execution.

OML4Py is available in the following Oracle database environments:

- The Python interpreter in Oracle Machine Learning Notebooks in your Oracle Autonomous Database. For more information, see Get Started with Notebooks for Data Analysis and Data Visualization in Using Oracle Machine Learning Notebooks.
  
  In this environment, all the required components are included, including Python, required Python libraries, and the Python interpreter in Notebooks.
- An OML4Py client connection to OML4Py in an on-premises Oracle Database instance.
  
  For this environment, you must install Python, the required Python libraries, and the OML4Py server components in the database, and you must install the OML4Py client. See Install OML4Py for On-Premises Databases.

Designed for problems involving both large and small volumes of data, OML4Py integrates Python with the database. With OML4Py, you can do the following:

- Develop, refine, and deploy user-defined Python functions and machine learning models that leverage the parallelism and scalability of the database to automate data preparation and machine learning.
- Run overloaded Python functions and use native Python syntax to manipulate in-database data, without having to learn SQL.
- Use Automated Machine Learning (AutoML) to enhance user productivity and machine learning results through automated algorithm and feature selection, as well as model tuning and selection.
- Use Embedded Python Execution to run user-defined Python functions in Python engines spawned and managed by the database environment. The user-defined functions and
data are automatically loaded to the engines as required, and when data-parallel and task-parallel execution is enabled.

Advantages of Oracle Machine Learning for Python

Using OML4Py to prepare and analyze data in or accessible to an Oracle database has many advantages for a Python user.

With OML4Py, you can do the following:

- **Operate on database data without using SQL**
  OML4Py transparently translates many standard Python functions into SQL. With OML4Py, you can create Python proxy objects that access, analyze, and manipulate data that resides in the database. OML4Py can automatically optimize the SQL by taking advantage of column indexes, query optimization, table partitioning, and database parallelism.
  OML4Py overloaded functions are available for many commonly used Python functions, including those on Pandas data frames for in-database execution.
  See Also: [Transparently Convert Python to SQL](#)

- **Automate common machine learning tasks**
  By using Oracle’s advanced Automated Machine Learning (AutoML) technology, both data scientists and beginner machine learning users can automate common machine learning modeling tasks such as algorithm selection and feature selection, and model tuning and selection, all of which leverage the parallel processing and scalability of the database.
  See Also: [About Automated Machine Learning](#)

- **Minimize data movement**
  By keeping the data in the database whenever possible, you eliminate the time involved in transferring the data to your client Python engine and the need to store the data locally. You also eliminate the need to manage the locally stored data, which includes tasks such as distributing the data files to the appropriate locations, synchronizing the data with changes that are made in the production database, and so on.
  See Also: [About Moving Data Between the Database and a Python Session](#)

- **Keep data secure**
  By keeping the data in the database, you have the security, scalability, reliability, and backup features of the database for managing the data.

- **Use the power of the database**
  By operating directly on data in the database, you can use the memory and processing power of the database and avoid the memory constraints of your client Python engine.

- **Use current data**
  As data is refreshed in the database, you have immediate access to current data.

- **Save Python objects to a datastore in the database**
  You can save Python objects to an OML4Py datastore for future use and for use by others.
**See Also:** About OML4Py Datastores

- **Build and store models in the database**
  Using Embedded Python Execution, you can build native Python models and store and manage them in an OML4Py datastore.
  
  You can also build in-database models, with, for example, an `oml` class such as the Decision Tree class `oml.dt`. These in-database models have proxy objects that reference the actual models. Keeping with normal Python behavior, when the Python engine terminates, all in-memory objects, including models, are lost. To prevent an in-database model created using OML4Py from being deleted when the database connection is terminated, you must store its proxy object in a datastore.

  **See Also:** About Machine Learning Classes and Algorithms

- **Score data**
  For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.
  
  For these OML4Py in-database models, you can also use the SQL `PREDICTION` function on the model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

- **Run user-defined Python functions in embedded Python engines**
  Using OML4Py Embedded Python Execution, you can store user-defined Python functions in the OML4Py script repository, and run those functions in Python engines spawned by the database environment. When a user-defined Python function runs, the database starts, controls, and manages one or more Python engines that can run in parallel. With the Embedded Python Execution functionality, you can do the following:

  - Use a select set of Python packages in user-defined functions that run in embedded Python engines
  - Use other Python packages in user-defined Python functions that run in embedded Python engines
  - Operationalize user-defined Python functions for use in production applications and eliminate porting Python code and models into other languages; avoid reinventing code to integrate Python results into existing applications
  - Seamlessly leverage your Oracle database as a high-performance computing environment for user-defined Python functions, providing data parallelism and resource management
  - Perform parallel simulations, for example, Monte Carlo analysis, using the `oml.index_apply` function
  - Generate PNG images and XML representations of both structured and image data, which can be used by Python clients and applications that use REST APIs.

  **See Also:** About Embedded Python Execution and the Script Repository

---

**Transparency Convert Python to SQL**

With the transparency layer classes, you can convert select Python objects to Oracle database objects and also invoke a range of familiar Python functions that are overloaded to invoke the corresponding SQL on tables in the database.
The OML4Py transparency layer does the following:

- Contains functions that convert Python pandas.DataFrame objects to database tables
- Overloads Python functions, translating their functionality into SQL
- Leverages proxy objects for database data
- Uses familiar Python syntax to manipulate database data

The following table lists the transparency layer functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.create</td>
<td>Creates a table in a the database schema from a Python data set.</td>
</tr>
<tr>
<td>oml_object.pull</td>
<td>Creates a local Python object that contains a copy of data referenced by the oml object.</td>
</tr>
<tr>
<td>oml.push</td>
<td>Pushes data from a Python session into an object in a database schema.</td>
</tr>
<tr>
<td>oml.sync</td>
<td>Creates a DataFrame proxy object in Python that represents a database table or view.</td>
</tr>
<tr>
<td>oml.dir</td>
<td>Return the names of oml objects in the Python session workspace.</td>
</tr>
<tr>
<td>oml.drop</td>
<td>Drops a persistent database table or view.</td>
</tr>
</tbody>
</table>

Transparency layer proxy classes map SQL data types or objects to corresponding Python types. The classes provide Python functions and operators that are the same as those on the mapped Python types. The following table lists the transparency layer data type classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.Boolean</td>
<td>A boolean series data class that represents a single column of 0, 1, and NULL values in database data.</td>
</tr>
<tr>
<td>oml.Bytes</td>
<td>A binary series data class that represents a single column of RAW or BLOB database data types.</td>
</tr>
<tr>
<td>oml.Float</td>
<td>A numeric series data class that represents a single column of NUMBER, BINARY_DOUBLE, or BINARY_FLOAT database data types.</td>
</tr>
<tr>
<td>oml.String</td>
<td>A character series data class that represents a single column of VARCHAR2, CHAR, or CLOB database data types.</td>
</tr>
</tbody>
</table>

The following table lists the mappings of OML4Py data types for both the reading and writing of data between Python and the database.
Table 1-3  Python and SQL Data Type Equivalencies

<table>
<thead>
<tr>
<th>Database Read</th>
<th>Python Data Types</th>
<th>Database Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Boolean</td>
<td>If oranumber == True, then NUMBER (the default), else BINARY_DOUBLE.</td>
</tr>
<tr>
<td>BLOB</td>
<td>bytes</td>
<td>BLOB</td>
</tr>
<tr>
<td>RAW</td>
<td></td>
<td>RAW</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>NUMBER</td>
<td>str</td>
<td>CHAR</td>
</tr>
<tr>
<td>CHAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOB</td>
<td></td>
<td>CLOB</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td></td>
<td>VARCHAR2</td>
</tr>
</tbody>
</table>

Manage Resources Used by OML4Py

If you find that your Python process is consuming too many of your machine's resources, or causing your machine to crash, you can get information about, or set limits for, the resources Python is using.

The Python system and process utilities library `psutil` is a cross-platform library for retrieving information on running processes and system utilization, such as CPU, memory, disks, network, and sensors, in Python. It is useful for system monitoring, profiling, limiting process resources, and the management of running processes.

The function `psutil.Process.rlimit` gets or sets process resource limits. In `psutil`, process resource limits are constants with names beginning with `psutil.RLIMIT_`. Each resource is controlled by a soft limit and hard limit tuple.

For example, `psutil.RLIMIT_AS` represents the maximum size (in bytes) of the virtual memory (address space) used by the process. The default limit of `psutil.RLIMIT_AS` can be `-1` (`psutil.RLIM_INFINITY`). You can lower the resource limit of `psutil.RLIMIT_AS` to prevent your Python program from loading too much data into memory, as shown in the following example.

**Example 1-1  Resource Control with psutil.RLIMIT_AS**

```python
import psutil
import numpy as np

# Get the current OS process.
p = psutil.Process()

# Get a list of available resources.
[attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']

# Display the Virtual Memory Size of the current process.
p.memory_info().vms

# Get the process resource limit RLIMIT_AS.
soft, hard = p.rlimit(psutil.RLIMIT_AS)
```
print('Original resource limits of RLIMIT_AS (soft/hard): {}\n{}\n'.format(soft, hard))

# Check the constant used to represent the limit for an unlimited resource.
pstyl.RLIM_INFINITY

# Set resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
pstyl.p.rlimit(pstyl.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = pstyl.p.rlimit(pstyl.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}\n{}\n'.format(cur_soft, cur_hard))

# Define a list of sizes to be allocated in MB (megabytes).
sz = [100, 200, 500, 1000]

# Define a megabyte variable in bytes.
MB = 1024*1024

# Allocate an increasing amount of data.
for val in sz:
  stmt = "Allocate %s MB " % val
try:
    print("virtual memory: %d MB" % int(pstyl.p.memory_info().vms/MB))
m = np.arange(val*MB/8, dtype="u8")
    print(stmt + " Success.")
except:
    print(stmt + " Fail.")
raise

# Delete the allocated variable.
del m

# Raise the soft limit of RLIMIT_AS to 2GB.
pstyl.p.rlimit(pstyl.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = pstyl.p.rlimit(pstyl.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}\n{}\n'.format(cur_soft, cur_hard))

# Retry: allocate an increasing amount of data.
for val in sz:
  stmt = "Allocate %s MB " % val
try:
    print("virtual memory: %d MB" % int(pstyl.p.memory_info().vms/MB))
m = np.arange(val*MB/8, dtype="u8")
    print(stmt + " Success.")
except:
    print(stmt + " Fail.")
raise
Listing for This Example

>>> import psutil
>>> import numpy as np

>>> # Get the current OS process.
... p = psutil.Process()

>>> # Get a list of available resources.
... [attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']
['RLIMIT_AS', 'RLIMIT_CORE', 'RLIMIT_CPU', 'RLIMIT_DATA',
 'RLIMITFSIZE', 'RLIMIT_LOCKS', 'RLIMIT_MEMLOCK', 'RLIMIT_MSGQUEUE',
 'RLIMIT_NICE', 'RLIMIT_NOFILE', 'RLIMIT_NPROC', 'RLIMIT_RSS',
 'RLIMIT_RTPRIO', 'RLIMIT_RTTIME', 'RLIMIT_SIGPENDING', 'RLIMIT_STACK']

>>> # Display the Virtual Memory Size of the current process.
... p.memory_info().vms
413175808

>>> # Get the process resource limit RLIMIT_AS.
... soft, hard = p.rlimit(psutil.RLIMIT_AS)

>>> print('Original resource limits of RLIMIT_AS (soft/hard): {} / {}'.format(soft, hard))
Original resource limits of RLIMIT_AS (soft/hard): -1/-1

>>> # Check the constant used to represent the limit for an unlimited resource.
... psutil.RLIM_INFINITY
-1

>>> # Set the resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))

>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)

>>> print('Current resource limits of RLIMIT_AS (soft/hard): {} / {}'.format(cur_soft, cur_hard))
Current resource limits of RLIMIT_AS (soft/hard): 1073741824/2147483648

>>> # Define a list of sizes to be allocated in MB (megabytes).
... sz = [100, 200, 500, 1000]

>>> # Define a megabyte variable in bytes.
... MB = 1024*1024

>>> # Allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise
virtual memory: 394 MB
Allocate 100 MB Success.
virtual memory: 494 MB
Allocate 200 MB Success.
virtual memory: 594 MB
Allocate 500 MB Fail.
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
MemoryError
>>> # Delete the allocated variable.
... del m
>>> # Raise the soft limit of RLIMIT_AS to 2GB.
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))
>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Current resource limits of RLIMIT_AS (soft/hard): {}/{}
Current resource limits of RLIMIT_AS (soft/hard): 2147483648/2147483648
>>> # Retry: allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise

virtual memory: 458 MB
Allocate 100 MB Success.
virtual memory: 558 MB
Allocate 200 MB Success.
virtual memory: 658 MB
Allocate 500 MB Success.
virtual memory: 958 MB
Allocate 1000 MB Success.

About the Python Components and Libraries in OML4Py

OML4Py requires an installation of Python, a number of Python libraries, as well as the OML4Py components.

- In Oracle Autonomous Database, OML4Py is already installed. The OML4Py installation includes Python, additional required Python libraries, and the OML4Py server components. A Python interpreter is included with Oracle Machine Learning Notebooks in Autonomous Database.
- You can install OML4Py in an on-premises Oracle Database. In this case, you must install Python, the additional required Python libraries, the OML4Py server
components, and an OML4Py client. See Install OML4Py for On-Premises Databases.

**Python Version in Current Release of OML4Py**

The current release of OML4Py is based on Python 3.9.5.

This version is in the current release of Oracle Autonomous Database. You must install it manually when installing OML4Py on an on-premises Oracle Database.

**Required Python Libraries**

The following Python libraries must be included.

- `cx_Oracle 8.1.0` *
- `cycler 0.10.0`
- `joblib 0.14.0`
- `kiwisolver 1.1.0`
- `matplotlib 3.1.2` *
- `numpy 1.18.1` *
- `pandas 0.25.3` *
- `pyparsing 2.4.0`
- `python-dateutil 2.8.1`
- `pytz 2019.3`
- `scikit-learn 0.23.1` *
- `scipy 1.6.1` *
- `six 1.13.0`
- `threadpoolctl 2.1.0`

All the above libraries are included with Python in the current release of Oracle Autonomous Database.

For an installation of OML4Py in an on-premises Oracle Database, you must install Python, which includes a number of the libraries, and you must additionally install the libraries marked with an asterisk (*) above. See Install OML4Py for On-Premises Databases.
Install OML4Py for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py with an on-premises Oracle database.

- **OML4Py On Premises System Requirements**
  Both client and server on-premises components are supported on the Linux platforms described in this topic.

- **Build and Install Python for Linux for On-Premises Databases**
  Instructions for installing Python for Linux for an on-premises Oracle database.

- **Install the Required Supporting Packages for Linux for On-Premises Databases**
  Installation instructions for the supporting Python packages for Linux for an on-premises Oracle database.

- **Install OML4Py Server for On-Premises Oracle Database**
  Instructions for installing and uninstalling the OML4Py server components for an on-premises Oracle database.

- **Install OML4Py Client for On-Premises Databases**
  Instructions for installing and uninstalling the on-premises OML4Py client.

**OML4Py On Premises System Requirements**

OML4Py on premises runs on 64-bit platforms only.

Both client and server on-premises components are supported on the Linux platforms listed in the table below.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Hardware Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Linux x86-64 7.x</td>
<td>Intel</td>
<td>64-bit Oracle Linux Release 7</td>
</tr>
<tr>
<td>Oracle Linux x86-64 8.x</td>
<td>Intel</td>
<td>64-bit Oracle Linux Release 8</td>
</tr>
</tbody>
</table>

**Table 2-2 On-Premises OML4Py Configuration Requirements and Server Support Matrix**

<table>
<thead>
<tr>
<th>Oracle Machine Learning for Python Version</th>
<th>Python Version</th>
<th>On-Premises Oracle Database Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>3.9.5</td>
<td>21c</td>
</tr>
</tbody>
</table>
Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises Oracle database.

Python 3.9.5 or a later version of 3.9 is required to install and use OML4Py.

These steps describe building and installing Python 3.9.5 for Linux.

1. Go to the Python website and download the Gzipped source tarball. The downloaded file name is Python-3.9.5.tgz.

2. Extract the contents to a directory of your choosing:
   tar -xvzf /path_to_downloaded_file/Python-3.9.5.tgz -C /path_to_extracted_files/

3. Go to the new directory:
   cd /path_to_extracted_files/Python-3.9.5/

4. OML4Py requires the presence of the libffi-devel, openssl-devel, tk-devel, xz-devel, and zlib-devel libraries.
   You can confirm that those libraries are present by issuing the following commands:

   ```
   rpm -qa libffi-devel
   rpm -qa openssl-devel
   rpm -qa tk-devel
   rpm -qa xz-devel
   rpm -qa zlib-devel
   ```

   If the libraries are present, then those commands should return messages such as the following. Depending on the version of Linux that you are using, such as version 7.3 or 7.5, the exact messages differ slightly.

   ```
   libffi-devel-3.0.13-19.el7.i686
   libffi-devel-3.0.13-19.el7.x86_64
   openssl-devel-1.0.2k-19.0.1.el7.x86_64
   tk-devel-8.5.13-6.el7.i686
   xz-devel-5.2.2-1.el7.x86_64
   zlib-devel-1.2.7-17.el7.x86_64
   zlib-devel-1.2.7-17.el7.i686
   ```

   The actual value returned depends on the version of Linux that you are using.
   If no output is returned, then install the packages as root.

   ```
   yum install libffi-devel
   yum install openssl-devel
   yum install tk-devel
   yum install xz-devel
   yum install zlib-devel
   ```
5. To build Python 3.9.5, enter the following commands, where `PREFIX` is the directory in which you installed Python-3.9.5:

```
Python-3.9.5$ ./configure --enable-shared --prefix=PREFIX
Python-3.9.5$ make clean; make
Python-3.9.5$ make altinstall
```

**Note:**

Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

6. Set environment variables: set `PYTHONHOME` and add it to your `PATH`, and set `LD_LIBRARY_PATH`:

```
export PYTHONHOME=PREFIX
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

You can now start Python by running the command `python3`.

If you run the command `python3` and you get the error `command not found`, then that means the system cannot find an executable named `python3` in `$PYTHONHOME/bin`. In that case, you need to create a symbolic link in your `PREFIX/bin` directory to link to your `python3.9` executable, which you can do with the following commands:

```
cd PREFIX/bin
ln -s python3.9 python3
```

## Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.

The on-premises OML4Py client requires the following Python packages:

- numpy 1.18.1
- pandas 0.25.3
- scipy 1.6.0
- matplotlib 3.1.2
- cx_Oracle 8.1.0
- scikit-learn 0.23.1
Use `pip3.9` to install the supporting packages. For each of the packages, run the following command, specifying the package and a proxy server:

```
pip3.9 install packagename --proxy="http://proxy.server:port"
```

For example, this command installs the `cx_Oracle` package using an example proxy server:

```
pip3.9 install cx_Oracle --proxy="http://www-proxy.example.com:80"
```

This command installs the `scikit-learn 0.23.1` package:

```
pip3.9 install scikit-learn==0.23.1
```

## Install OML4Py Server for On-Premises Oracle Database

The following instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle Database.

- Install OML4Py Server for Linux for On-Premises Oracle Database 21c
- Verify OML4Py Client Installation for On-Premises Databases
- Grant Users the Required Privileges for On-Premises Database
- Create New Users for On-Premises Oracle Database.
- Uninstall the OML4Py Client for On-Premises Databases

### Install OML4Py Server for Linux for On-Premises Oracle Database 21c

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 21c.

You can install OML4Py by using a Python script included in your 21c database or by using the Database Configuration Assistant (DBCA).

#### Install OML4Py By Using a Python Script

To install the OML4Py server for Linux for an on-premises Oracle Database 21c, run the server installation Python script `pyqcfg.sql`.

1. At your operating system prompt, start SQL*Plus and log in to your Oracle pluggable database (PDB) directly.
2. Run the `pyqcfg.sql` script. The script is under `$ORACLE_HOME/oml4py/server`.

To capture the log, spool the installation steps to an external file. The following example uses the PDB `PDB1` and gives example values for the script arguments.

```
$ sqlplus / as sysdba
SQL> spool install.txt
SQL> alter session set container=PDB1;
SQL> ALTER PROFILE DEFAULT LIMIT PASSWORD_VERIFY_FUNCTION NULL;
```
SQL> @$ORACLE_HOME/oml4py/server/pyqcfg.sql

define permtbl_value = SYSAUX --> Specify a permanent tablespace for the PYQSYS schema
define temptbl_value = TEMP --> Specify a temporary tablespace
define orahome_value = /u01/app/oracle/product/21.3.0.0/dbhome_1 --> Specify the ORACLE_HOME directory
define pythonhome = /opt/Python-3.9.5 --> Specify the PYTHON_HOME directory

3. Open the install.txt file to see if any errors occurred.

Install OML4Py With the Database Configuration Assistant (DBCA)

You can install OML4Py by using DBCA. For complete instruction on using DBCA, see Database Configuration Assistant Command Reference for Silent Mode.

The basic syntax to install OML4Py is:

dba -configureOML4PY

You can include the following parameters:

• -oml4pyConfigTablespace to configure the tablespace of the PYQSYS schema for OML4Py. The default tablespace is SYSAUX.
• -enableOml4pyEmbeddedExecution to enable the embedded Python component of Oracle Machine Learning for Python. The default value is TRUE.

Verify OML4Py Server Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

1. On the OML4Py server database instance, start SQL*Plus as the OML user.

   sqlplus oml_user/oml_user_password

2. Create a user-defined Python function and store it in the OML4Py script repository.

   BEGIN
   sys.pyqScriptCreate('TEST',
     'def func():return 1 + 1');
   END;
   /

   PL/SQL procedure successfully completed.
3. Run the user-defined Python function with the Embedded Python Execution function `pyqEval`. To use an Embedded Python Execution function, you must have installed the OML4Py server with Embedded Python Execution enabled.

   ```sql
   SELECT * FROM table(pyqEval(NULL, 'XML','TEST'));
   ```

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;root&gt;&lt;int&gt;2&lt;/int&gt;&lt;/root&gt;</td>
<td></td>
</tr>
</tbody>
</table>

4. In a Python client session, connect to the OML4Py server. Invoke the same user-defined function by name, using the `oml.do_eval` function. To use an Embedded Python Execution function, you must have installed the OML4Py client with Embedded Python Execution enabled. In the following example, replace the values for the parameters with those for your database.

   ```python
   import oml
   oml.connect(user='oml_user', password='oml_user_password',
               host='myhost',
               port=1521, sid='mysid')
   res = oml.do_eval(func='TEST')
   res
   ```

   2

5. When you are finished testing, you can drop the test.

   ```sql
   BEGIN
   sys.pyqScriptDrop('TEST');
   END;
   /
   ```

   PL/SQL procedure successfully completed.

---

Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

**User Privileges**

After installing the OML4Py server on an on-premises Oracle database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE
• CREATE VIEW  
• CREATE PROCEDURE  
• CREATE MINING MODEL  
• EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the oml.algo package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```

**Script Repository and Datastore Management**

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

• Store user-defined Python functions in the script repository.
• Drop user-defined Python function from the repository
• Grant or revoke permission to use a user-defined Python function in the script repository.
• Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```

**Create New Users for On-Premises Oracle Database.**

The `pyquser.sql` script is a convenient way to create a new OML4Py user for an on-premises database.

**About the pyquser.sql Script**

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The `sysdba` privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:
Create a New User

To use the pyquser.sql script, go the server subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the sysdba user to run the pyquser.sql script.

**Example 2-1  Creating New Users**

This example creates the user oml_user with the permanent tablespace USERS with an unlimited quota, the temporary tablespace TEMP, and grants the PYQADMIN role to the oml_user.

```sql
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

Enter value for password: <type your password>

This example creates the user oml_user2 with the permanent tablespace USERS with a 20 megabyte quota, the temporary tablespace TEMP, and without the PYQADMIN role.

```sql
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE
```

Enter value for password: <type your password>

Uninstall the OML4Py Server from an On-Premises Database

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle Database.

**Uninstall the On-Premises OML4Py Server for Linux**

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the PYTHONHOME environment variable is set to the Python3.8 directory.

```bash
echo $PYTHONHOME
```
2. Verify that "PYTHONPATH" environment variable is set to the directory in which the "oml modules are installed."

```bash
echo $PYTHONPATH
```

If it is not set to the proper directory, set it.

```bash
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

3. Change directories to the directory containing the server installation zip file.

```bash
cd $ORACLE_HOME/oml4py
```

4. Run the server installation Perl script with the "-u" argument.

```bash
perl -Iserver server/server.pl -u
```

When the script displays "Proceed?, enter y or yes."

### Install OML4Py Client for On-Premises Databases

Instructions for installing and uninstalling the on-premises OML4Py client.

### Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle database.

To connect the OML4Py client for Linux to an on-premises Oracle database, you must have Oracle Instant Client installed on your local system.

### Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle database. See the Oracle Support Note "Client / Server Interoperability Support Matrix for Different Oracle Versions (Doc ID 207303.1)."

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select Instant Client for Linux x86-64.
2. Locate the section for your version of Oracle Database. These instructions use the Version 18.5.0.0.0 section.
3. In the Base section, in the Download column, click the zip file for the Basic Package or Basic Light Package and save the file in an accessible directory on your system. These instructions use the directory /opt/oracle.

4. Go to the folder that you selected and unzip the package. For example:

   cd /opt/oracle
   unzip instantclient-basic-linux.x64-19.11.0.0.0dbru.zip

   Extracting the package creates the subdirectory instantclient_19_11, which contains the Oracle Instant Client files.

5. Install the libaio package with sudo or as the root user. (In some Linux distributions this package is called libaio1.) For example:

   sudo yum install libaio

6. Add the directory that contains the Oracle Instant Client files to the beginning of your LD_LIBRARY_PATH environment variable:

   export LD_LIBRARY_PATH=/opt/oracle/instantclient_19_11:
   $LD_LIBRARY_PATH

Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.

To use the OML4Py client to connect to an on-premises Oracle database, the following are required:

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and select Oracle Machine Learning for Python Downloads (v1.0).
   c. Select Oracle Machine Learning for Python Client Install for Oracle Database on Linux 64 bit.
d. Save the zip file to an accessible directory. These instructions use a directory named `oml4py`.

2. Go to the directory to which you downloaded the zip file and unzip the file.

   ```
cd oml4py
unzip oml4py-client-linux-x86-64-1.0.zip
```

The contents are extracted to a subdirectory named `client`, which contains these three files:

- `client.pl`
- `OML4PInstallShared.pm`
- `oml-1.0-cp38-cp38m-linux_x86_64.whl`

View the Optional Arguments to the Client Installation Perl Script

In the directory that contains the downloaded the installation zip file (`oml4py` in these instructions), run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```perl
perl -Iclient client/client.pl --help
```

Oracle Machine Learning for Python 1.0 Client.

Copyright (c) 2021 Oracle and/or its affiliates. All rights reserved.

Usage: client.pl [OPTION]...

- `i, --install`    install or upgrade (default)
- `u, --uninstall`  uninstall
- `y`               never prompt
- `--ask`           interactive mode (default)
- `--no-embed`      do not install embedded Python functionality
- `--no-deps`       turn off dependencies checking
- `--target <dir>`  install client into <dir>

By default, the installation script installs the Embedded Python Execution module. If you don't want to install the module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see Install the Required Supporting Packages for Linux for On-Premises Databases.

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:
1. In the directory that contains the extracted client installation Perl script, run the script. The following command runs the Perl script with the target directory specified:

   perl -Iclient client/client.pl --target path_to_target_dir

   When the script displays Proceed?, enter y or yes.

   If you use the --target <dir> argument to install the oml module to the specified directory, then add that location to environment variable PYTHONPATH so that Python can find the module:

   export PYTHONPATH=path_to_target_dir

2. To verify that oml modules are successfully installed and are ready to use, start Python and import oml. At the Linux prompt, enter python3.

   python3

   At the Python prompt, enter import oml

   import oml

3. Display the location of the installation directory.  
   If you didn't use the --target <dir> argument, then the installed oml modules are stored under $PYTHONHOME/lib/python3.9/site-packages/. Again, you must have write permission for the target directory.

   In Python, after importing the oml module, you can display the directory in which the client is installed. At the Python prompt, enter:

   oml.__path__

Connect to the OML4Py Server

Start Python, import oml, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses oml_user as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

   import oml
   oml.connect(user='oml_user', password='oml_user_password', host=myhost, port=1521, sid=mysid)

   After connecting, you can run any of the examples in this publication. For example, you could run Example 3-8.
Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

```python
import oml
oml.connect(user='oml_user', password='oml_user_password', host='myhost', port=1521, sid='mysid')
```

2. Create a user-defined Python function and store it in the OML4Py script repository.

```python
oml.script.create("TEST", func='def func():return 1 + 1', overwrite=TRUE)
```

3. Invoke the user-defined function, using the `oml.do_eval` function.

```python
import oml
res = oml.do_eval(func='TEST')
res
```

4. When you are finished testing, you can drop the test.

```python
oml.script.drop("TEST")
```

Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

**Uninstall the On-Premises OML4Py Client for Linux**

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays *Proceed?, enter y or yes.*
If the client is successfully uninstalled, you'll see the following message:

Uninstalling oml-1.0:
  Successfully uninstalled oml-1.0
Get Started with Oracle Machine Learning for Python

Learn how to use OML4Py in Oracle Machine Learning Notebooks and how to move data between the local Python session and the database.

These actions are described in the following topics.

- Use OML4Py with Oracle Autonomous Database
- Use OML4Py with an On-Premises Oracle Database
- Move Data Between the Database and a Python Session
- Save Python Objects in the Database

Use OML4Py with Oracle Autonomous Database

OML4Py is available through the Python interpreter in Oracle Machine Learning Notebooks in Oracle Autonomous Database.

For more information, see Get Started with Notebooks for Data Analysis and Data Visualization in Using Oracle Machine Learning Notebooks.

Use OML4Py with an On-Premises Oracle Database

After the OML4Py server and client components have been installed on your on-premises Oracle database server and you have installed the OML4Py client on your local system, you can connect your client Python session to the OML4Py server.

To connect an OML4Py client to an on-premises Oracle database, you first import the oml module and then connect as described in the following topics.

About Connecting to an On-Premises Oracle Database

OML4Py client components connect a Python session to the OML4Py server components on an on-premises Oracle database server.

The connection makes the data in an on-premises Oracle database schema available to the Python user. It also makes the processing power, memory, and storage capacities of the database server available to the Python session through the OML4Py client interface. To use that data and those capabilities, you must create a connection to the Oracle database server.

To use the Automatic Machine Learning (AutoML) capabilities of OML4Py, the following must be true:

- A connection pool must be running on the server.
- You must explicitly use the automl argument in an oml.connect invocation to specify the running connection pool on the server.
Note:

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```sql
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```sql
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

Note:

Because an AutoML connection requires more database resources than an `oml.connect` connection without AutoML does, you should create an AutoML connection only if you are going to use the AutoML classes.

Note:

- Only one type of connection can be active during a Python session: either a connection with AutoML enabled or one without it enabled. You can, however, terminate one type of connection and initiate the other type during the same Python session. Terminating either type of connection results in the automatic clean up of any temporary objects created in the session during that connection.

  If you want to save any objects that you created in one type of connection before changing to the other type, then save the objects in an OML4Py datastore before invoking `oml.connect` again. You can then reload the objects after reconnecting.

- The `oml.connect` function uses the `cx_Oracle` Python package for database connectivity. In some cases, you might want to use the `cx_Oracle.connect` function of that package to connect to a database. That function has advantages such as the following:
  
  - Allows multiple connections to a multiple databases, which might be useful in an running Embedded Python Execution functions
  
  - Permits some SQL data manipulation language (DML) operations that are not available in an `oml.connect` connection

  For information on the `cx_Oracle.connect` function, see Connecting to Oracle Database in the `cx_Oracle` documentation.

OML4Py Connection Functions

The OML4Py functions related to database connections are the following.
### About Oracle Wallets

An Oracle wallet is a secure software container that stores authentication and signing credentials for an Oracle Database.

You can create an OML4Py connection to an Oracle Database instance by specifying an Oracle wallet. For instructions on creating an Oracle wallet, see Managing the Secure External Password Store for Password Credentials in *Oracle Database Security Guide*.

The Oracle wallet must contain a credential that specifies a *tnsnames.ora* entry such as the following:

```plaintext
waltcon = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myserv.example.com)))
```

To be able to use an Oracle wallet to create an OML4Py connection in which you can use Automatic Machine Learning (AutoML), the wallet must also have a credential that has a *tnsnames.ora* entry for a server connection pool such as the following:

```plaintext
waltcon_pool = (DESCRIPTION= (ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))
(CONNECT_DATA=(SID=mysid)(SERVER=pooled)))
```

*Note:*

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```sql
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```sql
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

For examples of creating a connection using an Oracle wallet, see Example 3-6 and Example 3-7.
Connect to an Oracle Database

Establish an OML4Py connection to an on-premises Oracle database with `oml.connect`.

The `oml.connect` function establishes a connection to the user’s schema in an on-premises Oracle database.

The syntax of the `oml.connect` function is the following.

```python
oml.connect(user=None, password=None, host=None, port=None, sid=None, service_name=None, dsn=None, encoding='UTF-8', nencoding='UTF-8', automl=None)
```

To create a basic connection to the database, you can specify arguments to the `oml.connect` function in the following mutually exclusive combinations:

- `user`, `password`, `dsn`
- `user`, `password`, `host`, `port`, `sid`
- `user`, `password`, `host`, `port`, `service_name`

The arguments specify the following values.

**Table 3-2 Parameters to `oml.connect`**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A string specifying a username.</td>
</tr>
<tr>
<td>password</td>
<td>A string specifying the password for the user.</td>
</tr>
<tr>
<td>host</td>
<td>A string specifying the name of the host machine on which the OML4Py server is installed.</td>
</tr>
<tr>
<td>port</td>
<td>An int or a string specifying the Oracle database port number on the host machine.</td>
</tr>
<tr>
<td>sid</td>
<td>A string specifying the system identifier (SID) of the Oracle database.</td>
</tr>
<tr>
<td>service_name</td>
<td>A string specifying the service name of the Oracle database.</td>
</tr>
<tr>
<td>dsn</td>
<td>A string specifying a data source name, which can be a TNS entry for the database or a TNS alias in an Oracle Wallet.</td>
</tr>
<tr>
<td>encoding</td>
<td>A string specifying the encoding to use for regular database strings.</td>
</tr>
<tr>
<td>nencoding</td>
<td>A string specifying the encoding to use for national character set database strings.</td>
</tr>
<tr>
<td>automl</td>
<td>A string or a boolean specifying whether to enable an Automatic Machine Learning (AutoML) connection, which uses the database-resident connection pool. If there is a connection pool running for a host, port, SID (or service name), then you can specify that host, port, SID (or service name) and automl=True. If the dsn argument is a data source name, then the automl argument must be a data source name for a running connection pool. If the dsn argument is a TNS alias, then the automl argument must be a TNS alias for a connection pool specified in an Oracle Wallet.</td>
</tr>
</tbody>
</table>

To use the AutoML capabilities of OML4Py, the following must be true:
• A connection pool must be running on the server.
• You must explicitly use the `automl` argument in an `oml.connect` invocation to specify the running connection pool on the server.

**Note:**

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```sql
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```sql
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

Only one active OML4Py connection can exist at a time during a Python session. If you call `oml.connect` when an active connection already exists, then the `oml.disconnect` function is implicitly invoked, any temporary objects that you created in the previous session are discarded, and the new connection is established. Before attempting to connect, you can discover whether an active connection exists by using the `oml.isconnected` function.

You explicitly end a connection with the `oml.disconnect` function. If you do not invoke `oml.disconnect`, then the connection is automatically terminated when the Python session ends.

**Examples**

In the following examples, the values of the some of the arguments to the `oml.connect` function are string variables that are not declared in the example. To use any of the following examples, replace the username, password, port, and variable argument values with the values for your user and database.

**Example 3-1  Connecting with a Host, Port, and SID**

This example uses the `host`, `port`, and `sid` arguments. It also shows the use of the `oml.isconnected`, `oml.check_embed`, and `oml.disconnect` functions.

```python
import oml

oml.connect(user='oml_user', password='oml_user_password', host='myhost', port=1521, sid='mysid')

# Verify that the connection exists.
oml.isconnected()

# Find out whether Embedded Python Execution is enabled in the database instance.
oml.check_embed()

# Disconnect from the database.
oml.disconnect()
```
# Verify that the connection has been terminated.

oml.isconnected()

**Listing for This Example**

```python
>>> import oml

>>> oml.connect(user='oml_user', password='oml_user_password', host='myhost',
... port=1521, sid='mysid')

>>> # Verify that the connection exists.
... oml.isconnected()
True

>>> # Find out whether Embedded Python Execution is enabled in the
... # database instance.
... oml.check_embed()
True

>>> # Disconnect from the database.
... oml.disconnect()

>>> # Verify that the connection has been terminated.
... oml.isconnected()
False
```

**Example 3-2    Connecting with Host, Port, and Service Name**

This example uses the **host**, **port**, and **service_name** arguments.

```python
import oml

oml.connect(user='oml_user', password='oml_user_password',
host='myhost',
            port=1521, service_name='myservice')
```

**Example 3-3    Connecting with a DSN Containing a SID**

This example uses the **dsn** argument to specify a SID.

```python
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))\"" "(CONNECT_DATA=(SID=mysid)))"

oml.connect(user='oml_user', password='oml_user_password', dsn=mydsn)
```

**Example 3-4    Connecting with a DSN Containing a Service Name**

This example uses the **dsn** argument to specify a service name.

```python
import oml
```
Example 3-5  Creating a Connection with a DSN and with AutoML Enabled

This example creates an OML4Py connection with AutoML enabled. The example connects to a local database.

```python
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myhost)\(PORT=1521))\(CONNECT_DATA=(SID=mysid)))"

dsn_pool = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)\(PORT=1521))\(CONNECT_DATA=(SERVICE_NAME=myservice.example.com)\(SERVER=POOLED)))"

oml.connect(user='oml_user', password='oml_user_password', dsn=mydsn)

# Verify that the connection exists and that AutoML is enabled.
oml.isconnected(check_automl=True)
```

Example 3-6  Connecting with an Oracle Wallet

This example creates a connection using the `dsn` argument to specify an Oracle wallet. The `dsn` value, `waltcon` in the example, must refer to the alias in the database `tnsnames.ora` file that was used to create the appropriate credential in the wallet.

```python
import oml

oml.connect(user='', password='', dsn='waltcon')
```

See Also:

About Oracle Wallets

Example 3-7  Connecting with an Oracle Wallet with AutoML Enabled

This example connects using an Oracle wallet to establish a connection with AutoML enabled by using the `dsn` and `automl` arguments. The example then verifies that the connection has AutoML enabled. The `dsn` and `automl` values, `waltcon` and `waltcon_pool` in the example, must refer to aliases in the database `tnsnames.ora` file that were used to create the appropriate credentials in the wallet.

```python
import oml
```
Move Data Between the Database and a Python Session

With OML4Py functions, you can interact with data structures in a database schema.

In your Python session, you can move data to and from the database and create temporary or persistent database tables. The OML4Py functions that perform these actions are described in the following topics.

- **About Moving Data Between the Database and a Python Session**
- **Push Local Python Data to the Database**
- **Pull Data from the Database to a Local Python Session**
- **Create a Python Proxy Object for a Database Object**
- **Create a Persistent Database Table from a Python Data Set**

### About Moving Data Between the Database and a Python Session

Using the functions described in this topic, you can move data between the your local Python session and an Oracle database schema.

The following functions create proxy `oml` Python objects from database objects, create database tables from Python objects, list the objects in the workspace, and drop tables and views.

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oml.create</code></td>
<td>Creates a persistent database table from a Python data set.</td>
</tr>
<tr>
<td><code>oml.cursor</code></td>
<td>Returns a <code>cx_Oracle cursor</code> object for the current OML4Py database connection.</td>
</tr>
<tr>
<td><code>oml.dir</code></td>
<td>Returns the names of the <code>oml</code> objects in the workspace.</td>
</tr>
<tr>
<td><code>oml.drop</code></td>
<td>Drops a persistent database table or view.</td>
</tr>
<tr>
<td><code>oml_object.pull</code></td>
<td>Creates a local Python object that contains a copy of the database data referenced by the <code>oml</code> object.</td>
</tr>
<tr>
<td><code>oml.push</code></td>
<td>Pushes data from the OML Notebooks Python session memory into a temporary table in the database.</td>
</tr>
<tr>
<td><code>oml.sync</code></td>
<td>Creates an <code>oml.DataFrame</code> proxy object in Python that represents a database table, view, or query.</td>
</tr>
</tbody>
</table>

With the `oml.push` function, you can create a temporary database table, and its corresponding proxy `oml.DataFrame` object, from a Python object in your OML Notebooks Python session. The temporary table is automatically deleted when the OML Notebook connection to the database ends unless you have saved its proxy object to a datastore before disconnecting.

With the `oml.push` function, you can create a temporary database table, and its corresponding proxy `oml.DataFrame` object, from a Python object in your local Python session. The temporary table is automatically deleted when the OML Notebook or OML4Py client connection to the database ends unless you have saved its proxy object to a datastore before disconnecting.
With the pull method of an oml object, you can create a local Python object that contains a copy of the database data represented by an oml proxy object.

The oml.push function implicitly coerces Python data types to oml data types and the pull method on oml objects coerces oml data types to Python data types.

With the oml.create function, you can create a persistent database table and a corresponding oml.DataFrame proxy object from a Python data set.

With the oml.sync function, you can synchronize the metadata of a database table or view with the oml object representing the database object.

With the oml.cursor function, you can create a cx_Oracle cursor object for the current database connection. You can use the cursor to run queries against the database, as shown in Example 3-13.

Related Topics
• Transparently Convert Python to SQL
  With the transparency layer classes, you can convert select Python objects to Oracle database objects and also invoke a range of familiar Python functions that are overloaded to invoke the corresponding SQL on tables in the database.

Push Local Python Data to the Database

Use the oml.push function to push data from your local Python session to a temporary table in your Oracle database schema.

The oml.push function creates a temporary table in the user’s database schema and inserts data into the table. It also creates and returns a corresponding proxy oml.DataFrame object that references the table in the Python session. The table exists as long as an oml object exists that references it, either in the Python session memory or in an OML4Py datastore.

The syntax of the oml.push function is the following:

```
oml.push(x, oranumber=True, dbtypes=None)
```

The x argument may be a pandas.DataFrame or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to COL1, COL2, and so on.

The SQL data types of the columns are determined by the following:

• OML4Py determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type.
  If the values in a column are all None, or if a column has inconsistent data types that are not None in the sampled rows, then a default column type cannot be determined and a ValueError is raised unless a SQL type for the column is specified by the dbtypes argument.

• For numeric columns, the oranumber argument, which is a bool, determines the SQL data type. If True (the default), then the SQL data type is NUMBER. If False, then the data type is BINARY_DOUBLE.
  If the data in x contains NaN values, then you should set oranumber to False.

• For string columns, the default type is VARCHAR2(4000).
• For bytes columns, the default type is BLOB.

With the dbtypes argument, you can specify the SQL data types for the table columns. The values of dbtypes may be either a dict that maps str to str values or a list of str values. For a dict, the keys are the names of the columns.

Example 3-8 Pushing Data to a Database Table

This example creates pd_df, a pandas.core.frame.DataFrame object with columns of various data types. It pushes pd_df to a temporary database table, which creates the oml_df object, which references the table. It then pulls the data from the oml_df object to the df object in local memory.

```python
import oml
import pandas as pd

pd_df = pd.DataFrame(
    {'numeric': [1, 1.4, -4, 3.145, 5, None],
     'string': [None, None, 'a', 'a', 'a', 'b'],
     'bytes': [b'a', b'b', b'c', b'c', b'd', b'e']})

# Push the data set to a database table with the specified dbtypes
# for each column.
oml_df = oml.push(pd_df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                      'string': 'CHAR(1)',
                                      'bytes': 'RAW(1)'})

# Display the data type of oml_df.
type(oml_df)

# Pull the data from oml_df into local memory.
df = oml_df.pull()

type(df)

# Create a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
       (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]

# Create an oml.DataFrame using the list.
oml_df2 = oml.push(lst, dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])
type(oml_df2)
```

Listing for This Example

```bash
>>> import oml
>>> import pandas as pd
...
>>> pd_df = pd.DataFrame(
...    {'numeric': [1, 1.4, -4, 3.145, 5, None],
...     'string': [None, None, 'a', 'a', 'a', 'b'],
...     'bytes': [b'a', b'b', b'c', b'c', b'd', b'e']})
>>> # Push the data set to a database table with the specified dbtypes
```
Pull Data from the Database to a Local Python Session

Use the `pull` method of an `oml` proxy object to create a Python object in your local Python session.

The `pull` method of an `oml` object returns a Python object of the same type. The object contains a copy of the database data referenced by the `oml` object. The Python object exists in-memory in the Python session in your OML Notebooks session.

The `pull` method of an `oml` object returns a Python object of the same type. The object contains a copy of the database data referenced by the `oml` object. The Python object exists in-memory in the Python session in OML Notebooks or in your OML4Py client Python session.

**Note:**

You can pull data to a local `pandas.DataFrame` only if the data can fit into the local Python session memory. Also, even if the data fits in memory but is still very large, you may not be able to perform many, or any, Python functions in the local Python session.
Example 3-9  Pulling Data into Local Memory

This example loads the iris data set and creates the IRIS database table and the
oml_iris proxy object that references that table. It displays the type of the oml_iris
object, then pulls the data from it to the iris object in local memory and displays its
type.

import oml
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

oml_iris = oml.create(iris, table = 'IRIS')

# Display the data type of oml_iris.
type(oml_iris)

# Pull the data from oml_iris into local memory.
iris = oml_iris.pull()

# Display the data type of iris.
type(iris)

# Drop the IRIS database table.
oml.drop('IRIS')

Listing for This Example

>>> import oml
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
>>> iris = datasets.load_iris()

>>> oml_iris = oml.create(iris, table = 'IRIS')

>>> # Display the data type of oml_iris.
... type(oml_iris)
... <class 'oml.core.frame.DataFrame'>

>>> # Pull the data from oml_iris into local memory.
... iris = oml_iris.pull()

>>> # Display the data type of iris.
... type(iris)
... <class 'pandas.core.frame.DataFrame'>

>>> # Drop the IRIS database table.
... oml.drop('IRIS')
Create a Python Proxy Object for a Database Object

Use the `oml.sync` function to create a Python object as a proxy for a database table, view, or SQL statement.

The `oml.sync` function returns an `oml.DataFrame` object or a dictionary of `oml.DataFrame` objects. The `oml.DataFrame` object returned by `oml.sync` is a proxy for the database object.

You can use the proxy `oml.DataFrame` object to select data from the table. When you run a Python function that selects data from the table, the function returns the current data from the database object. However, if some application has added a column to the table, or has otherwise changed the metadata of the database object, the `oml.DataFrame` proxy object does not reflect such a change until you again invoke `oml.sync` for the database object.

**Tip:**

To conserve memory resources and save time, you should only create proxies for the tables that you want to use in your Python session.

You can use the `oml.dir` function to list the `oml.DataFrame` proxy objects in the environment for a schema.

The syntax of the `oml.sync` function is the following:

```
oml.sync(schema=None, regex_match=False, table=None, view=None, query=None)
```

With the `schema` argument, you can specify the schema in which to create a Python environment and proxy objects. Only one environment for a given database schema can exist at a time. If `schema=None`, then objects are created in the current user’s schema.

To create an `oml.DataFrame` object for a table, use the `table` parameter. To create one for a view, use the `view` parameter. To create one for a SQL `SELECT` statement, use the `query` parameter. You can only specify one of these parameters in an `oml.sync` invocation: the argument for one of the parameters must be a string and the argument for each of the other two parameters must be `None`.

Creating a proxy object for a query enables you to create an `oml.DataFrame` object without creating a view in the database. This can be useful when you do not have the `CREATE VIEW` system privilege for the current schema. You cannot use the `schema` parameter and the `query` parameter in the same `oml.sync` invocation.

With the `regex_match` argument, you can specify whether the value of the `table` or `view` argument is a regular expression. If `regex_match=True`, then `oml.sync` creates `oml.DataFrame` objects for each database object that matches the pattern. The matched tables or views are returned in a `dict` with the table or view names as keys.
Example 3-10  Creating a Python Object for a Database Table

This example creates an oml.DataFrame Python object as a proxy for a database table. For this example, the table COFFEE exists in the user's schema.

```python
import oml

# Create the Python object oml_coffee as a proxy for the database table COFFEE.
oml_coffee = oml.sync(table = 'COFFEE')
type(oml_coffee)

# List the proxy objects in the schema.
oml.dir()

oml_coffee.head()
```

Listing for This Example

```python
>>> import oml

>>> # Create the Python object oml_coffee as a proxy for the database table COFFEE.
... oml_coffee = oml.sync(table = 'COFFEE')
>>> type(oml_coffee)
<class 'oml.core.frame.DataFrame'>

>>> # List the proxy objects in the schema.
... oml.dir()
['oml_coffee']

>>> oml_coffee.head()
   ID  COFFEE  WINDOW
 0   1   esp    w
 1   2   cap    d
 2   3   cap    w
 3   4   kon    w
 4   5   ice    w
```

Example 3-11  Using the regex_match Argument

This example uses the regex_match argument in creating a dict object that contains oml.DataFrame proxy objects for tables whose names start with C. For this example, the COFFEE and COLOR tables exist in the user's schema and are the only tables whose names start with C.

```python
# Create a dict of oml.DataFrame proxy objects for tables whose names start with 'C'.
oml_cdat = oml.sync(table="^C", regex_match=True)

oml_cdat.keys()

oml_cdat['COFFEE'].columns

oml_cdat['COLOR'].columns
```
Listing for This Example

```python
>>> # Create a dict of oml.DataFrame proxy objects for tables
... # whose names start with 'C'.
... oml_cdat = oml.sync(table="^C", regex_match=True)

>>> oml_cdat.keys()
dict_keys(['COFFEE', 'COLOR'])

>>> oml_cdat['COFFEE'].columns
['ID', 'COFFEE', 'WINDOW']

>>> oml_cdat['COLOR'].columns
['REGION', 'EYES', 'HAIR', 'COUNT']
```

Example 3-12  Synchronizing an Updated Table

This example uses `oml.sync` to create an `oml.DataFrame` for the database table COFFEE. For the example, the new column BREW has been added to the database table by some other database process after the first invocation of `oml.sync`. Invoking `oml.sync` again synchronizes the metadata of the `oml.DataFrame` with those of the table.

```python
oml_coffee = oml.sync(table = "COFFEE")

# After a new column has been inserted into the table.

oml_coffee = oml.sync(table = "COFFEE")
```

Create a Persistent Database Table from a Python Data Set

Use the `oml.create` function to create a persistent table in your database schema from data in your Python session.

The `oml.create` function creates a table in the database schema and returns an `oml.DataFrame` object that is a proxy for the table. The proxy `oml.DataFrame` object has the same name as the table.
Note:

When creating a table in Oracle Machine Learning for Python, if you use lowercase or mixed case for the name of the table, then you must use the same lowercase or mixed case name in double quotation marks when using the table in a SQL query or function. If, instead, you use an all uppercase name when creating the table, then the table name is case-insensitive: you can use uppercase, lowercase, or mixed case when using the table without using double quotation marks. The same is true for naming columns in a table.

You can delete the persistent table in a database schema with the `oml.drop` function.

Caution:

Use the `oml.drop` function to delete a persistent database table. Use the `del` statement to remove an `oml.DataFrame` proxy object and its associated temporary table; `del` does not delete a persistent table.

The syntax of the `oml.create` function is the following:

```python
oml.create(x, table, oranumber=True, dbtypes=None, append=False)
```

The `x` argument is a `pandas.DataFrame` or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to COL1, COL2, and so on. The `table` argument is a string that specifies a name for the table.

The SQL data types of the columns are determined by the following:

- **OML4Py** determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type.
  - If the values in a column are all `None`, or if a column has inconsistent data types that are not `None` in the sampled rows, then a default column type cannot be determined and a `ValueError` is raised unless a SQL type for the column is specified by the `dbtypes` argument.
  - For numeric columns, the `oranumber` argument, which is a `bool`, determines the SQL data type. If `True` (the default), then the SQL data type is NUMBER. If `False`, then the data type is BINARY DOUBLE.
  - If the data in `x` contains NaN values, then you should set `oranumber` to `False`.
  - For string columns, the default type is VARCHAR2(4000).
  - For bytes columns, the default type is BLOB.

With the `dbtypes` parameter, you can specify the SQL data types for the table columns. The values of `dbtypes` may be either a `dict` that maps str to str values or a list of str values. For a `dict`, the keys are the names of the columns. The `dbtypes` parameter is ignored if the `append` argument is `True`. 
The `append` argument is a `bool` that specifies whether to append the `x` data to an existing table.

**Example 3-13 Creating Database Tables from a Python Data Set**

This example creates a cursor object for the database connection, creates a `pandas.core.frame.DataFrame` with columns of various data types, then creates a series of tables using different `oml.create` parameters and shows the SQL data types of the table columns.

```python
import oml

# Create a cursor object for the current OML4Py database
# connection to run queries and get information from the database.
cr = oml.cursor()

import pandas as pd

df = pd.DataFrame({
  'numeric': [1, 1.4, -4, 3.145, 5, 2],
  'string': [None, None, 'a', 'a', 'a', 'b'],
  'bytes': [b'a', b'b', b'c', b'c', b'd', b'e']}})

# Get the order of the columns
df.columns

# Create a table with the default parameters.
oml_df1 = oml.create(df, table = 'tbl1')

# Show the default SQL data types of the columns.
_=cr.execute("select data_type from all_tab_columns where table_name = 'tbl1'")
cr.fetchall()

# Create a table with `oranumber` set to `False`.
oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)

# Show the SQL data typea of the columns.
_=cr.execute("select data_type from all_tab_columns where table_name = 'tbl2'")
cr.fetchall()

# Create a table with `dbtypes` specified as a dict mapping column names
# to SQL data types.
oml_df3 = oml.create(df, table = 'tbl3',
    dbtypes = {'numeric': 'BINARY_DOUBLE',
               'bytes': 'RAW(1)'})

# Show the SQL data types of the columns.
_=cr.execute("select data_type from all_tab_columns where table_name = 'tbl3'")
cr.fetchall()

# Create a table with `dbtypes` specified as a list of SQL data types
# matching the order of the columns.
oml_df4 = oml.create(df, table = 'tbl4',
    dbtypes = ['BINARY_DOUBLE', 'RAW(1)'])
```

---

Chapter 3
Move Data Between the Database and a Python Session

3-17
dbtypes = ['BINARY_DOUBLE','VARCHAR2','RAW(1)'])

# Show the SQL data type of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name = 'tbl4'")
cr.fetchall()

# Create a table from a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
      (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
oml_df5 = oml.create(lst, table = 'tbl5',
                     dbtypes = ['BINARY_DOUBLE','CHAR(1)','RAW(1)'])

# Close the cursor
cr.close()

# Drop the tables.
oml.drop('tbl1')
oml.drop('tbl2')
oml.drop('tbl3')
oml.drop('tbl4')
oml.drop('tbl5')

Listing for This Example

>>> import oml
>>> # Create a cursor object for the current OML4Py database
... # connection to run queries and get information from the database.
... cr = oml.cursor()
>>> import pandas as pd
>>> df = pd.DataFrame({"numeric": [1, 1.4, -4, 3.145, 5, 2],
...                    "string" : [None, None, 'a', 'a', 'a', 'b'],
...                    "bytes" : [b'a', b'b', b'c', b'c', b'd', b'e']})
>>> # Get the order of the columns.
... df.columns
Index(['numeric', 'string', 'bytes'], dtype='object')
>>>
>>> # Create a table with the default parameters.
... oml_df1 = oml.create(df, table = 'tbl1')
>>> # Show the default SQL data types of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name = 'tbl1'")
... cr.fetchall()
[('NUMBER',), ('VARCHAR2',), ('BLOB',)]
>>> # Create a table with oranumber set to False.
... oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)
>>> # Show the SQL data types of the columns.
Save Python Objects in the Database

You can save Python objects in OML4Py datastores, which persist in the database.
You can grant or revoke read privilege access to a datastore or its objects to one or more users. You can restore the saved objects in another Python session.

The following topics describe the OML4Py functions for creating and managing datastores:

- About OML4Py Datastores
- Save Objects to a Datastore
- Load Saved Objects From a Datastore
- Get Information About Datastores
- Get Information About Datastore Objects
- Delete Datastore Objects
- Manage Access to Stored Objects

**About OML4Py Datastores**

In an OML4Py datastore, you can store Python objects, which you can then use in subsequent Python sessions; you can also make them available to other users or programs.

Python objects, including OML4Py proxy objects, exist only for the duration of the current Python session unless you explicitly save them. You can save a Python object, including `oml.proxy` objects, to a named datastore and then load that object in a later Python session, including an Embedded Python Execution session. OML4Py creates the datastore in the user’s database schema. A datastore, and the objects it contains, persist in the database until you delete them.

You can grant or revoke read privilege permission to another user to a datastore that you created or to objects in a datastore.

OML4Py has Python functions for managing objects in a datastore. It also has PL/SQL procedures for granting or revoking the read privilege and database views for listing available datastores and their contents.

Using a datastore, you can do the following:

- Save OML4Py and other Python objects that you create in one Python session and load them in another Python session.
- Pass arguments to Python functions for use in Embedded Python Execution.
- Pass objects for use in Embedded Python Execution. You could, for example, use the `oml glm` class to build an Oracle Machine Learning model and save it in a datastore. You could then use that model to score data in the database through Embedded Python Execution.

**Python Interface for Datastores**

The following table lists the Python functions for saving and managing objects in a datastore.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oml.ds.delete</code></td>
<td>Deletes one or more datastores or Python objects from a datastore.</td>
</tr>
<tr>
<td><code>oml.ds.dir</code></td>
<td>Lists the datastores available to the current user.</td>
</tr>
</tbody>
</table>
The following table lists the Python functions for managing access to datastores and datastore objects.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.ds.load</td>
<td>Loads Python objects from a datastore into the user’s session.</td>
</tr>
<tr>
<td>oml.ds.save</td>
<td>Saves Python objects to a named datastore in the user’s database schema.</td>
</tr>
</tbody>
</table>

Save Objects to a Datastore

The `oml.ds.save` function saves one or more Python objects to a datastore.

OML4Py creates the datastore in the current user’s schema.

The syntax of `oml.ds.save` is the following:

```python
oml.ds.save(objs, name, description=' ', grantable=None, overwrite=False, append=False, compression=False)
```

The `objs` argument is a dict that contains the name and object pairs to save to the datastore specified by the `name` argument.

With the `description` argument, you can provide some descriptive text that appears when you get information about the datastore. The `description` parameter has no effect when used with the `append` parameter.

With the `granteble` argument, you can specify whether the read privilege to the datastore may be granted to other users.

If you set the `overwrite` argument to `TRUE`, then you can replace an existing datastore with another datastore of the same name.

If you set the `append` argument to `TRUE`, then you can add objects to an existing datastore. The `overwrite` and `append` arguments are mutually exclusive.

If you set compression to `True`, then the serialized Python objects are compressed in the datastore.
Example 3-14  Saving Python Objects to a Datastore

This example demonstrates creating datastores.

```python
import oml
from sklearn import datasets
from sklearn import linear_model
import pandas as pd

# Load three data sets and create oml.DataFrame objects for them.
wine = datasets.load_wine()
x = pd.DataFrame(wine.data, columns = wine.feature_names)
y = pd.DataFrame(wine.target, columns = ['Class'])

# Create the database table WINE.
ml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
ml_wine.columns

diabetes = datasets.load_diabetes()
x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
ml_diabetes = oml.create(pd.concat([x, y], axis=1),
                          table = 'DIABETES')
mml_diabetes.columns

boston = datasets.load_boston()
x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
y = pd.DataFrame(boston.target, columns = ['Value'])
ml_boston = oml.create(pd.concat([x, y], axis=1), table = 'BOSTON')
mml_boston.columns

# Save the wine Bunch object to the datastore directly,
# along with the oml.DataFrame proxy object for the BOSTON table.
oml.ds.save(objs={'wine':wine, 'ml_boston':ml_boston},
            name='ds_data', description = 'wine datasets')

# Save the oml_diabetes proxy object to an existing datastore.
oml.ds.save(objs={'ml_diabetes':ml_diabetes},
            name='ds_data', append=True)

# Save the oml_wine proxy object to another datastore.
oml.ds.save(objs={'ml_wine':ml_wine},
            name='ds_data', description = 'wine dataset')

# Create regression models using sklearn and oml.
# The regr1 linear model is a native Python object.
regr1 = linear_model.LinearRegression()
regr1.fit(boston.data, boston.target)

# The regr2 GLM model is an oml object.
regr2 = oml.glm("regression")
X = ml_boston.drop('Value')
y = ml_boston['Value']
regr2 = regr2.fit(X, y)

# Save the native Python object and the oml proxy object to a datastore
```

Chapter 3
Save Python Objects in the Database

3-22
# and allow the read privilege to be granted to them.
ooml.ds.save(objs={"regr1":regr1, 'regr2':regr2},
    name="ds_pymodel", grantable=True)

# Grant the read privilege to the datastore to every user.
ooml.grant(name="ds_pymodel", typ="datastore", user=None)

# List the datastores to which the read privilege has been granted.
ooml.ds.dir(dstype="grant")

Listing for This Example

```python
>>> import oml
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>> import pandas as pd

>>> # Load three data sets and create oml.DataFrame objects for them.
>>> wine = datasets.load_wine()
>>> x = pd.DataFrame(wine.data, columns = wine.feature_names)
>>> y = pd.DataFrame(wine.target, columns = ['Class'])

>>> # Create the database table WINE.
... oml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
>>> oml_wine.columns
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline', 'Class']

>>> diabetes = datasets.load_diabetes()
>>> x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
>>> y = pd.DataFrame(diabetes.target, columns=['disease_progression'])

>>> oml_diabetes = oml.create(pd.concat([x, y], axis=1),
...                           table = "DIABETES")
>>> oml_diabetes.columns
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6',
 'disease_progression']

>>> boston = datasets.load_boston()
>>> x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
>>> y = pd.DataFrame(boston.target, columns = ['Value'])

>>> oml_boston = oml.create(pd.concat([x, y], axis=1), table = "BOSTON")
>>> oml_boston.columns
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
 'PTRATIO', 'B', 'LSTAT', 'Value']

>>> # Save the wine Bunch object to the datastore directly,
... # along with the oml.DataFrame proxy object for the BOSTON table.
... oml.ds.save(objs={'wine':wine, 'oml_boston':oml_boston},
...     name="ds_pydata", description = "python datasets")

>>> # Save the oml_diabetes proxy object to an existing
... datastore.
... oml.ds.save(objs={'oml_diabetes':oml_diabetes},
...     name="ds_pydata", description = "python datasets")
```
>>> oml.ds.save(objs={©oml_wine©:oml_wine},
... name="ds_wine_data", description = "wine dataset")

>>> # Create regression models using sklearn and oml.
... # The regr1 linear model is a native Python object.
... regr1 = linear_model.LinearRegression()
>>> regr1.fit(boston.data, boston.target)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

>>> # The regr2 GLM model is an oml proxy object.
... regr2 = oml.glm("regression")
>>> X = oml_boston.drop('Value')
>>> y = oml_boston['Value']
>>> regr2 = regr2.fit(X, y)

>>> # Save the native Python object and the oml proxy object to a datastore
... # and allow the read privilege to be granted to them.
... oml.ds.save(objs={©regr1©:regr1, ©regr2©:regr2},
... name="ds_pymodel", grantable=True)

>>> # Grant the read privilege to the ds_pymodel datastore to every user.
... oml.grant(name="ds_pymodel", typ="datastore", user=None)

>>> # List the datastores to which the read privilege has been granted.
... oml.ds.dir(dstype="grant")
<table>
<thead>
<tr>
<th>datastore_name</th>
<th>grantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_pymodel</td>
<td>PUBLIC</td>
</tr>
</tbody>
</table>

## Load Saved Objects From a Datastore

The `oml.ds.load` function loads one or more Python objects from a datastore into a Python session.

The syntax of `oml.ds.load` is the following:

oml.ds.load(name, objs=None, owner=None, to_globals=True)

The `name` argument specifies the datastore that contains the objects to load.

With the `objs` argument, you identify a specific object or a list of objects to load.

With the boolean `to_globals` parameter, you can specify whether the objects are loaded to a global workspace or to a dictionary object. If the argument to `to_globals` is `True`, then `oml.ds.load` function loads the objects into the global workspace. If the argument is `False`, then the function returns a `dict` object that contains pairs of object names and values.
The `oml.ds.load` function raises a `ValueError` if the `name` argument is an empty string or if the owner of the datastore is not the current user and the read privilege for the datastore has not been granted to the current user.

**Example 3-15   Loading Objects from Datastores**

This example loads objects from datastores. For the creation of the datastores used in this example, see Example 3-14.

```python
import oml

# Load all Python objects from a datastore to the global workspace.
sorted(oml.ds.load(name="ds_pydata"))

# Load the named Python object from the datastore to the global workspace.
oml.ds.load(name="ds_pymodel", objs=["regr2"])

# Load the named Python object from the datastore to the user's workspace.
oml.ds.load(name="ds_pymodel", objs=["regr1"], toGlobals=False)
```

**Listing for This Example**

```python
>>> import oml

>>> # Load all Python objects from a datastore to the current workspace.
... sorted(oml.ds.load(name="ds_pydata"))
['oml_boston', 'oml_diabetes', 'wine']

>>> # Load the named Python object from the datastore to the global workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr2"])
['regr2']

>>> # Load the named Python object from the datastore to the user's workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr1"], toGlobals=False)
{'regr1': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)}
```

**Get Information About Datastores**

The `oml.ds.dir` function provides information about datastores.

The syntax of `oml.ds.dir` is the following:

```python
oml.ds.dir(name=None, regex_match=False, dstype='user')
```

Use the `name` parameter to get information about a specific datastore.

Optionally, you can use the `regex_match` and `dstype` parameters to get information about datastores with certain characteristics. The valid arguments for `dstype` are the following:
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Lists all of the datastores to which the current user has the read privilege.</td>
</tr>
<tr>
<td>grant</td>
<td>Lists the datastores for which the current user has granted read privilege to other users.</td>
</tr>
<tr>
<td>granted</td>
<td>Lists the datastores for which other users have granted read privilege to the current user.</td>
</tr>
<tr>
<td>grantable</td>
<td>Lists the datastores that the current user can grant the read privilege to.</td>
</tr>
<tr>
<td>user</td>
<td>Lists the datastores created by current user.</td>
</tr>
<tr>
<td>private</td>
<td>Lists the datastores that the current user cannot grant the read privileges to.</td>
</tr>
</tbody>
</table>

The `oml.ds.dir` function returns a `pandas.DataFrame` object that contains different columns depending on which `dstype` argument you use. The following table lists the arguments and the columns returned for the values supplied.

<table>
<thead>
<tr>
<th>dstype Argument</th>
<th>Columns in the DataFrame Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>DSNAME, which contains the datastore name</td>
</tr>
<tr>
<td>private</td>
<td>NOBJ, which contains the number of objects in the datastore</td>
</tr>
<tr>
<td>grantable</td>
<td>DSIZE, which contains the size in bytes of each object in the datastore</td>
</tr>
<tr>
<td></td>
<td>CDATE, which contains the creation date of the datastore</td>
</tr>
<tr>
<td></td>
<td>DESCRIPTION, which contains the optional description of the datastore</td>
</tr>
<tr>
<td>all</td>
<td>All of the columns returned by the <code>user</code>, <code>private</code>, and <code>grantable</code> values, plus this additional column:</td>
</tr>
<tr>
<td></td>
<td>DSOWNER, which contains the owner of the datastore</td>
</tr>
<tr>
<td>granted</td>
<td>DSNAME, which contains the datastore name</td>
</tr>
<tr>
<td></td>
<td>GRANTEE, which contains the name of the user to which the read privilege to the datastore has been granted by the current session user</td>
</tr>
</tbody>
</table>

Example 3-16  Getting Information About Datastores

This example demonstrates using different combinations of arguments to the `oml.ds.dir` function. It demonstrates using `oml.dir` to list some or all of the datastores. For the creation of the datastores used in this example, see Example 3-14.

```python
import oml

# Show all saved datastores.
oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]

# Show datastores to which other users have been granted the read privilege.
oml.ds.dir(dstype="grant")

# Show datastores whose names match a pattern.
```
Get Information About Datastore Objects

The `oml.ds.describe` function provides information about the objects in a datastore. The syntax of `oml.ds.describe` is the following:

```python
oml.ds.describe(name, owner=None)
```

The `name` argument is a string that specifies the name of a datastore. The `owner` argument is a string that specifies the owner of the datastore or `None` (the default). If you do not specify the owner, then the function returns information about the datastore if it is owned by the current user.

The `oml.ds.describe` function returns a `pandas.DataFrame` object, each row of which represents an object in the datastore. The columns of the `DataFrame` are the following:

- `object_name`, which specifies the name of the object
- `class`, which specifies the class of the object
- `size`, which specifies the size of the object in bytes
- `length`, which specifies the length of the object
- `row_count`, which specifies the rows of the object
- `col_count`, which specifies the columns of the object

This function raises a `ValueError` if the following occur:
• The current user is not the owner of the datastore and has not been granted read privilege for the datastore.

• The datastore does not exist.

Example 3-17     Getting Information About Datastore Objects

This example demonstrates the using the `oml.ds.describe` function. For the creation of the datastore used in this example, see Example 3-14.

```python
import oml

# Describe the contents of the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')
```

Listing for This Example

```python
>>> import oml

>>> # Describe the contents of the ds_pydata datastore.
... oml.ds.describe(name='ds_pydata')
  object_name    class          size  length  row_count  col_count
0  oml_boston   oml.DataFrame   1073   506      506         14
1  oml_diabetes oml.DataFrame   964   442      442         11
2      wine       Bunch      24177    5       1          5
```

Delete Datastore Objects

The `oml.ds.delete` function deletes datastores or objects in a datastore.

Use the `oml.ds.delete` function to delete one or more datastores in your database schema or to delete objects in a datastore.

The syntax of `oml.ds.delete` is the following:

```python
oml.ds.delete(name, objs=None, regex_match=False)
```

The argument to the `name` parameter may be one of the following:

- A string that specifies the name of the datastore to modify or delete, or a regular expression that matches the datastores to delete.
- A list of `str` objects that name the datastores from which to delete objects.

The `objs` parameter specifies the objects to delete from a datastore. The argument to the `objs` parameter may be one of the following:

- A string that specifies the object to delete from one or more datastores, or a regular expression that matches the objects to delete.
None (the default), which deletes the entire datastore or datastores.

The regex_match parameter is a bool that indicates whether the name or objs arguments are regular expressions. The default value is False. The regex_match parameter operates as follows:

- If regex_match=False and if name is not None, and:
  - If objs=None, then oml.ds.delete deletes the datastore or datastores specified in the name argument.
  - If you specify one or more datastores with the name argument and one or more datastore objects with the objs argument, then oml.ds.delete deletes the specified Python objects from the datastores.

- If regex_match=True and:
  - If objs=None, then oml.ds.delete deletes the datastores you specified in the name argument.
  - If the name argument is a string and you specify one or more datastore objects with the objs argument, then oml.ds.delete deletes from the datastore the objects whose names match the regular expression specified in the objs argument.
  - If the name argument is a list of str objects, then the objs argument must be a list of str objects of the same length as name, and oml.ds.delete deletes from the datastores the objects whose names match the regular expressions specified in objs.

This function raises an error if the following occur:

- A specified datastore does not exist.
- Argument regex_match is False and argument name is a list of str objects larger than 1 and argument objs is not None.
- Argument regex_match is True and arguments name and objs are lists that are not the same length.

**Example 3-18 Deleting Datastore Objects**

This example demonstrates the using the oml.ds.delete function. For the creation of the datastores used in this example, see Example 3-14.

```python
import oml

# Show the existing datastores.
oml.ds.dir()

# Show the Python objects in the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')

# Delete some objects from the datastore.
oml.ds.delete(name='ds_pydata', objs=['wine', "oml_boston"])

# Delete a datastore.
oml.ds.delete(name='ds_pydata')

# Delete all datastores whose names match a pattern.
oml.ds.delete(name='_pymodel', regex_match=True)
```
# Show the existing datastores again.

```python
oml.ds.dir()
```

### Listing for This Example

```python
>>> import oml

>>> # Show the existing datastores.
... oml.ds.dir()

datastore_name  object_count  size                date   description
datasets
0   ds_pydata             3  26214 2019-05-18 21:04:06  python datasets
1   ds_pymodel             2   6370 2019-05-18 21:08:18 None
2   ds_wine_data             1  1410 2019-05-18 21:06:53  wine dataset

>>> # Show the Python objects in the ds_pydata datastore.
... oml.ds.describe(name="ds_pydata")

object_name          class  size  length  row_count  col_count
0    oml_boston  oml.DataFrame   1073     506        506         14
1  oml_diabetes  oml.DataFrame    964     442        442         11
2          wine          Bunch  24177       5          1          5

>>> # Delete some objects from a datastore.
... oml.ds.delete(name="ds_pydata", objs=['wine', 'oml_boston'])

>>> # Delete a datastore.
... oml.ds.delete(name="ds_pydata")

'ds_pydata'

>>> # Delete all datastores whose names match a pattern.
... oml.ds.delete(name="_pymodel", regex_match=True)

['ds_pymodel']

>>> # Show the existing datastores again.
... oml.ds.dir()

datastore_name  object_count  size                date   description
0   ds_wine_data             1  1410 2019-05-18 21:06:53  wine dataset
```

## Manage Access to Stored Objects

The `oml.grant` and `oml.revoke` functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

The `oml.grant` function grants the read privilege to another user to a datastore or to a user-defined Python function in the OML4Py script repository. The `oml.revoke` function revokes that privilege.
The syntax of these functions is the following:

```python
oml.grant(name, typ='datastore', user=None)
oml.revoke(name, typ='datastore', user=None)
```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository or the name of a datastore.

The `typ` parameter must be specified. The argument is a string that is either `datastore` or `pyqscript`.

The `user` argument is a string that specifies the user to whom read privilege to the named datastore or user-defined Python function is granted or from whom it is revoked, or `None` (the default). If you specify `None`, then the read privilege is granted to or revoked from all users.

**Example 3-19  Granting and Revoking Access to Datastores**

This example displays the datastores to which the read privilege has been granted to all users. It revokes read privilege from the `ds_pymodel` datastore and displays the datastores with public read privilege again. It next grants the read privilege to the user `SH` and finally displays once more the datastores to which read privilege has been granted. For the creation of the datastores used in this example, see Example 3-14.

```python
import oml

# Show datastores to which other users have been granted read privilege.
oml.ds.dir(dstype="grant")

# Revoke the read privilege from every user.
oml.revoke(name="ds_pymodel", typ="datastore", user=None)

# Again show datastores to which read privilege has been granted.
oml.ds.dir(dstype="grant")

# Grant the read privilege to the user SH.
oml.grant(name="ds_pymodel", typ="datastore", user="SH")

oml.ds.dir(dstype="grant")
```

**Listing for This Example**

```python
>>> import oml

>>> # Show datastores to which other users have been granted read privilege.
... oml.ds.dir(dstype="grant")
  datastore_name grantee
  0   ds_pymodel   PUBLIC
>>> 
>>> # Revoke the read privilege from every user.
... oml.revoke(name="ds_pymodel", typ="datastore", user=None)
>>> 
>>> # Again show datastores to which read privilege has been granted to
... other users.
... oml.ds.dir(dstype="grant")
Empty DataFrame
```
Example 3-20  Granting and Revoking Access to User-Defined Python Functions

This example grants the read privilege to the MYLM user-defined Python function to the user SH and then revokes that privilege. For the creation of the user-defined Python functions used in this example, see Example 7-6.

Listing for This Example

```python
>>> # List the user-defined Python functions available only to the current user.
oml.script.dir(sctype='user')

# Grant the read privilege to the MYLM user-defined Python function to the user SH.
oml.grant(name="MYLM", typ="pyqscript", user="SH")

# List the user-defined Python functions to which read privilege has been granted.
oml.script.dir(sctype="grant")

# Revoke the read privilege to the MYLM user-defined Python function from the user SH.
oml.revoke(name="MYLM", typ="pyqscript", user="SH")

# List the granted user-defined Python functions again to see if the revocation was successful.
oml.script.dir(sctype="grant")
```
>>> # Revoke the read privilege to the MYLM user-defined Python function from the user SH.
... oml.revoke(name="MYLM", typ="pyqscript", user="SH")

>>> # List the granted user-defined Python functions again to see if the revocation was successful.
... oml.script.dir(sctype="grant")
Empty DataFrame
Columns: [name, grantee]
Index: []

Related Topics

- **Save Objects to a Datastore**
  The `oml.ds.save` function saves one or more Python objects to a datastore.

- **Create and Store a User-Defined Python Function**
  Use the `oml.script.create` function to add a user-defined Python function to the script repository.
Prepare and Explore Data

Use OML4Py methods to prepare data for analysis and to perform exploratory analysis of the data.

Methods of the OML4Py data type classes make it easier for you to prepare very large enterprise database-resident data for modeling. These methods are described in the following topics.

- Prepare Data
- Explore Data
- Render Graphics

Prepare Data

Using methods of OML4Py data type classes, you can prepare data for analysis in the database, as described in the following topics.

About Preparing Data in the Database

OML4Py data type classes have methods that enable you to use Python to prepare database data for analysis.

You can perform data preparation operations on large quantities of data in the database and then continue operating on that data in-database or pull a subset of the results to your local Python session where, for example, you can use third-party Python packages to perform other operations.

The following table lists methods with which you can perform common data preparation tasks and indicates whether the OML4Py data type class supports the method.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Appends another oml data object of the same class to an oml object.</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ceil</td>
<td>Computes the ceiling of each element in an oml.Float series data object.</td>
<td>×</td>
<td>×</td>
<td>✔️</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>concat</td>
<td>Combines an oml data object column-wise with one or more other data objects.</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Table 4-1  (Cont.) Methods Supported by Data Types

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>count_pattern</td>
<td>Counts the number of occurrences of a pattern in each string.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>create_view</td>
<td>Creates an Oracle Database view for the data represented by the OML4Py data object.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dot</td>
<td>Calculates the inner product of the current oml.Float object with another oml.Float, or does matrix multiplication with an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>drop</td>
<td>Drops specified columns in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>drop_duplicates</td>
<td>Removes duplicated elements from an oml series data object or duplicated rows from an oml.DataFrame.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dropna</td>
<td>Removes missing elements from an oml series data object, or rows containing missing values from an oml.DataFrame.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>exp</td>
<td>Computes element-wise e to the power of values in an oml.Float series data object.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>find</td>
<td>Finds the lowest index in each string in which a substring is found that is greater than or equal to a start index.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>floor</td>
<td>Computes the floor of each element in an oml.Float series data object.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>head</td>
<td>Returns the first $n$ elements of an oml series data object or the first $n$ rows of an oml.DataFrame.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KFold</td>
<td>Splits the oml data object randomly into $k$ consecutive folds.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>len</td>
<td>Computes the length of each string in an oml.Bytes or oml.String series data object.</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>log</td>
<td>Calculates an element-wise logarithm, to the given base, of values in the oml.Float series data object.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>materialize</td>
<td>Pushes the contents represented by an OML4Py proxy object (a view, a table, and so on) into a table in Oracle Database.</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>merge</td>
<td>Joins another oml.DataFrame to an oml.DataFrame.</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>replace</td>
<td>Replaces an existing value with another value.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>rename</td>
<td>Renames columns of an oml.DataFrame.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>round</td>
<td>Rounds oml.Float values to the specified decimal place.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>select_types</td>
<td>Returns the subset of columns that are included or excluded based on their oml data type.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>split</td>
<td>Splits an oml data object randomly into multiple sets.</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>sqrt</td>
<td>Computes the square root of each element in an oml.Float series data object.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>tail</td>
<td>Returns the last $n$ elements of an oml series data object or the last $n$ rows of an oml.DataFrame.</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>

**Select Data**

A typical step in preparing data for analysis is selecting or filtering values of interest from a larger data set.

The examples in this section demonstrate selecting data from an oml.DataFrame object by rows, by columns, and by value.

The examples use the oml_iris object created by the following code, which imports the sklearn.datasets package and loads the iris data set. It creates the $x$ and $y$ variables, and
then creates the persistent database table IRIS and the oml.DataFrame object oml.iris as a proxy for the table.

import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2: 'virginica'}[x], iris.target)), columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

The examples are in the following topics:

- Select the First or Last Number of Rows
- Select Data by Column
- Select Data by Value

Select the First or Last Number of Rows

The head and tail methods return the first or last number of elements.

The default number of rows selected is 5.

Example 4-1 Selecting the First and Last Number of Rows

This example selects rows from the oml.DataFrame object oml_iris. It displays the first five rows and ten rows of oml_iris and then the last five and ten rows.

```python
# Display the first 5 rows.
oml_iris.head()

# Display the first 10 rows.
oml_iris.head(10)

# Display the last 5 rows.
oml_iris.tail()

# Display the last 10 rows.
oml_iris.tail(10)
```

Listing for This Example

```python
>>> # Display the first 5 rows.
... oml_iris.head()

     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0      5.1        3.5           1.4          0.2    setosa
1      4.9        3.0           1.4          0.2    setosa
```
Select Data by Column

Example 4-2  Selecting Data by Columns

The example selects two columns from oml_iris and creates the oml.DataFrame object iris_projected1 with them. It then displays the first three rows of iris_projected1. The example also selects a range of columns from oml_iris, creates iris_projected2, and displays its first three rows. Finally, the example selects columns from oml_iris by data types, creates iris_projected3, and displays its first three rows.

# Select all rows with the specified column names.
iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
iris_projected1.head(3)
# Select all rows with columns whose indices are in the range [1, 4).
iris_projected2 = oml_iris[:, 1:4]
iris_projected2.head(3)

# Select all rows with columns of oml.String data type.
iris_projected3 = oml_iris.select_types(include=[oml.String])
iris_projected3.head(3)

Listing for This Example

```python
>>> # Select all rows with specified column names.
... iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
>>> iris_projected1.head(3)
      Sepal_Length  Petal_Length
0          5.1           1.4
1          4.9           1.4
2          4.7           1.3

>>> # Select all rows with columns whose indices are in range [1, 4).
... iris_projected2 = oml_iris[:, 1:4]
>>> iris_projected2.head(3)
      Sepal_Width  Petal_Length  Petal_Width
0          3.5           1.4          0.2
1          3.0           1.4          0.2
2          3.2           1.3          0.2

>>> # Select all rows with columns of oml.String data type.
... iris_projected3 = oml_iris.select_types(include=[oml.String])
>>> iris_projected3.head(3)
        Species
0        setosa
1        setosa
2        setosa
```

Select Data by Value

Example 4-3  Selecting Data by Value

This example filters `oml_iris` to produce `iris_of_filtered1`, which contains the values from the rows of `oml_iris` that have a petal length of less than 1.5 and that are in the Sepal_Length and Petal_Length columns. The example also filters the data using conditions, so that `oml_iris_filtered2` contains the values from `oml_iris` that have a petal length of less than 1.5 or a sepal length equal to 5.0 and `oml_iris_filtered3` contains the values from `oml_iris` that have a petal length of less than 1.5 and a sepal length larger than 5.0.

```python
# Select sepal length and petal length where petal length
# is less than 1.5.
oml_iris_filtered1 = oml_iris[oml_iris["Petal_Length"] < 1.5,
                             ["Sepal_Length", "Petal_Length"]]
len(oml_iris_filtered1)
oml_iris_filtered1.head(3)
```
### Using the AND and OR conditions in filtering.

# Select all rows in which petal length is less than 1.5 or sepal length
# sepal length is 5.0.
ml_iris_filtered2 = ml_iris[(ml_iris["Petal_Length"] < 1.5) |
                          (ml_iris["Sepal_Length"] == 5.0), :]

len(ml_iris_filtered2)
ml_iris_filtered2.head(3)

# Select all rows in which petal length is less than 1.5 and
# sepal length is larger than 5.0.
ml_iris_filtered3 = ml_iris[(ml_iris["Petal_Length"] < 1.5) &
                          (ml_iris["Sepal_Length"] > 5.0), :]

len(ml_iris_filtered3)
ml_iris_filtered3.head()
Combine Data

You can join data from `oml.DataFrame` objects that represent database tables by using the `append`, `concat`, and `merge` methods.

Examples of using these methods are in the following topics.

- Append Data from One Object to Another Object
- Combine Two Objects
- Join Data From Two Objects

Append Data from One Object to Another Object

Use the `append` method to join two objects of the same data type.

**Example 4-4   Appending Data from Two Tables**

This example first appends the `oml.Float` series object `num1` to another `oml.Float` series object, `num2`. It then appends an `oml.DataFrame` object to another `oml.DataFrame` object, which has the same column types.

```python
import oml
import pandas as pd

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                   "val" : ["a", "b", "c", "d", "e"],
                   "ch" : ["p", "q", "r", "a", "b"],
                   "num" : [4, 3, 6.7, 7.2, 5]})

oml_df = oml.push(df)

# Append an oml.Float series object to another.
num1 = oml_df['id']
num2 = oml_df['num']
num1.append(num2)

# Append an oml.DataFrame object to another.
x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
x.append(y)
```

**Listing for This Example**

```bash
>>> import oml
>>> import pandas as pd

>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                    "val" : ["a", "b", "c", "d", "e"],
...                    "ch" : ["p", "q", "r", "a", "b"],
...                    "num" : [4, 3, 6.7, 7.2, 5]})

>>> oml_df = oml.push(df)

>>> oml_df[['id', 'val']].append(oml_df[['num', 'ch']].append(oml_df[['id', 'val']]))
```
>>> # Append an oml.Float series object to another.
... num1 = oml_df['id']
>>> num2 = oml_df['num']
>>> num1.append(num2)
[1, 2, 3, 4, 5, 4, 3, 6.7, 7.2, 5]

>>> # Append an oml.DataFrame object to another.
... x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
>>> y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
>>> x.append(y)

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>5.0</td>
</tr>
<tr>
<td>5</td>
<td>4.0</td>
</tr>
<tr>
<td>6</td>
<td>3.0</td>
</tr>
<tr>
<td>7</td>
<td>6.7</td>
</tr>
<tr>
<td>8</td>
<td>7.2</td>
</tr>
<tr>
<td>9</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Combine Two Objects

Use the `concat` method to combine columns from one object with those of another object. The `auto_name` argument of the `concat` method controls whether to invoke automatic name conflict resolution. You can also perform customized renaming by passing in a dictionary mapping strings to objects.

To combine two objects with the `concat` method, both objects must represent data from the same underlying database table, view, or query.

Example 4-5 Combining Data Column-Wise

This example first combines the two `oml.DataFrame` objects `x` and `y` column-wise. It then concatenates object `y` with the `oml.Float` series object `w`.

```python
import oml
import pandas as pd
from collections import OrderedDict

df = pd.DataFrame({'id': [1, 2, 3, 4, 5],
                   'val': ['a', 'b', 'c', 'd', 'e'],
                   'ch': ['p', 'q', 'r', 'a', 'b'],
                   'num': [4, 3, 6.7, 7.2, 5]})

oml_df = oml.push(df)

# Create two oml.DataFrame objects and combine the objects column-wise.
x = oml_df[['id', 'val']]  # 1st column oml.Float, 2nd column oml.String
y = oml_df[['num', 'ch']]  # 1st column oml.Float, 2nd column oml.String
x.concat(y)

# Create an oml.Float object with the rounded exponential of two times
# the values in the num column of the oml_df object, then
# concatenate it with the oml.DataFrame object y using a new column name.
w = (oml_df['num'] * 2).exp().round(decimals=2)
```
y.concat({©round(exp(2*num))©:w})

# Concatenate object x with multiple objects and turn on automatic
# name conflict resolution.
z = oml_df[:, 'id']
x.concat([z, w, y], auto_name=True)

# Concatenate multiple oml data objects and perform customized renaming.
x.concat(OrderedDict([(©ID©,z), (©round(exp(2*num))©,w), (©New_©,y)]))

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict

>>> df = pd.DataFrame({
...                   'id': [1, 2, 3, 4, 5],
...                   'val': ['a', 'b', 'c', 'd', 'e'],
...                   'ch': ['p', 'q', 'r', 'a', 'b'],
...                   'num': [4, 3, 6.7, 7.2, 5]
...                })
>>> oml_df = oml.push(df)

>>> # Create two oml.DataFrame objects and combine the objects column-wise.
... x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>> x.concat(y)
   id  val  num  ch
0  1    a  4.0  p
1  2    b  3.0  q
2  3    c  6.7  r
3  4    d  7.2  a
4  5    e  5.0  b

>>> # Create an oml.Float object with the rounded exponential of two times
... # the values in the num column of the oml_df object, then
... # concatenate it with the oml.DataFrame object y using a new column name.
... w = (oml_df['num']*2).exp().round(decimals=2)
>>> y.concat({©round(exp(2*num))©:w})
   num  ch  round(exp(2*num))
0  4.0  p  2980.96
1  3.0  q  403.43
2  6.7  r  660003.22
3  7.2  a 1794074.77
4  5.0  b  22026.47

>>> # Concatenate object x with multiple objects and turn on automatic
... # name conflict resolution.
... z = oml_df[:, 'id']
>>> x.concat([z, w, y], auto_name=True)
   id  val  id3  num  num5  ch
0  1    a    1  2980.96  4.0  p
```
Join Data From Two Objects

Use the merge method to join data from two objects.

Example 4-6  Joining Data from Two Tables

This example first performs a cross join on the oml.DataFrame objects $x$ and $y$, which creates the oml.DataFrame object $xy$. The example performs a left outer join on the first four rows of $x$ with the oml.DataFrame object other on the shared column id and applies the suffixes .l and .r to column names on the left and right side, respectively. The example then performs a right outer join on the id column on the left side object $x$ and the num column on the right side object $y$.

```python
import oml
import pandas as pd

df = pd.DataFrame({'id' : [1, 2, 3, 4, 5],
                  'val' : ['a', 'b', 'c', 'd', 'e'],
                  'ch' : ['p', 'q', 'r', 'a', 'b'],
                  'num' : [4, 3, 6.7, 7.2, 5]})

oml_df = oml.push(df)
x = oml_df[['id', 'val']]  
y = oml_df[['num', 'ch']]  

# Perform a cross join.
xy = x.merge(y)
xy

# Perform a left outer join.
x.head(4).merge(other=oml_df[['id', 'num']], on='id',
                suffixes=['.l','.r'])

# Perform a right outer join.
x.merge(other=y, left_on='id', right_on='num', how='right')
```

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
```
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                   "val" : ["a", "b", "c", "d", "e"],
...                   "ch" : ["p", "q", "r", "a", "b"],
...                   "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)
>>> x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>> # Perform a cross join.
... xy = x.merge(y)
... xy
... id_l val_l  num_r ch_r
0   1   a    4.0   p
1   1   a    3.0   q
2   1   a    6.7   r
3   1   a    7.2   a
4   1   a    5.0   b
5   2   b    4.0   p
6   2   b    3.0   q
7   2   b    6.7   r
8   2   b    7.2   a
9   2   b    5.0   b
10  3   c    4.0   p
11  3   c    3.0   q
12  3   c    6.7   r
13  3   c    7.2   a
14  3   c    5.0   b
15  4   d    4.0   p
16  4   d    3.0   q
17  4   d    6.7   r
18  4   d    7.2   a
19  4   d    5.0   b
20  5   e    4.0   p
21  5   e    3.0   q
22  5   e    6.7   r
23  5   e    7.2   a
24  5   e    5.0   b
>>> # Perform a left outer join.
... x.head(4).merge(other=oml_df[['id', 'num']], on="id",
...                  suffixes=[".l", ".r"])
... id val.l  num.r
0   1   a    4.0
1   2   b    3.0
2   3   c    6.7
3   4   d    7.2
>>> # Perform a right outer join.
... x.merge(other=y, left_on="id", right_on="num", how="right")
... id_l val_l  num_r ch_r
0   3.0  c    3.0   q
1   4.0  d    4.0   p
2   5.0  e    5.0   b
Clean Data

In preparing data for analysis, a typical step is to transform data by dropping some values. You can filter out unneeded data by using the `drop`, `drop_duplicates`, and `dropna` methods.

**Example 4-7 Filtering Data**

This example demonstrates ways of dropping columns with the `drop` method, dropping missing values with the `dropna` method, and dropping duplicate values with the `drop_duplicates` method.

```python
import pandas as pd
import oml

df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None, None],
                   'string1': [None, None, 'a', 'a', 'a', 'b', None],
                   'string2': ['x', None, 'z', 'z', 'z', 'x', None]})

oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                'string1': 'CHAR(1)',
                                'string2': 'CHAR(1)'})

# Drop rows with any missing values.
oml_df.dropna(how='any')

# Drop rows in which all column values are missing.
oml_df.dropna(how='all')

# Drop rows in which any numeric column values are missing.
oml_df.dropna(how='any', subset=['numeric'])

# Drop duplicate rows.
oml_df.drop_duplicates()

# Drop rows that have the same value in column 'string1' and 'string2'.
oml_df.drop_duplicates(subset=['string1', 'string2'])

# Drop column 'string2'
oml_df.drop('string2')

Listing for This Example

```
```python
>>> # Drop rows with any missing values.
... oml_df.dropna(how='any')
  numeric  string1  string2
0    -4.000      a       z
1    -4.000      a       z
2     5.432      a       z

>>> # Drop rows in which all column values are missing.
... oml_df.dropna(how='all')
  numeric  string1  string2
0     1.000    None       x
1     1.400    None      None
2    -4.000      a       z
3    -4.000      a       z
4     5.432      a       z
5      NaN       b       x

>>> # Drop rows in which any numeric column values are missing.
... oml_df.dropna(how='any', subset=['numeric'])
  numeric  string1  string2
0     1.000    None       x
1     1.400    None      None
2    -4.000      a       z
3    -4.000      a       z
4     5.432      a       z

>>> # Drop duplicate rows.
... oml_df.drop_duplicates()
  numeric  string1  string2
0     5.432      a       z
1     1.000    None       x
2    -4.000      a       z
3     NaN       b       x
4     1.400    None      None
5     NaN       None      None

>>> # Drop rows that have the same value in columns 'string1' and 'string2'.
... oml_df.drop_duplicates(subset=['string1', 'string2'])
  numeric  string1  string2
0    -4.0      a       z
1     1.4      None      None
2     1.0      None      x
3     NaN      b       x

>>> # Drop the column 'string2'.
... oml_df.drop('string2')
  numeric  string1
0     1.000    None
1     1.400    None
2    -4.000      a
3    -4.000      a
4     5.432      a
```
Split Data

Sample and randomly partition data with the split and KFold methods.

In analyzing large data sets, a typical operation is to randomly partition the data set into subsets for training and testing purposes, which you can do with these methods. You can also sample data with the split method.

**Example 4-8  Splitting Data into Multiple Sets**

This example demonstrates splitting data into multiple sets and into k consecutive folds, which can be used for k-fold cross-validation.

```python
import oml
import pandas as pd
from sklearn import datasets

digits = datasets.load_digits()
pd_digits = pd.DataFrame(digits.data,
                         columns=['IMG'+str(i) for i in range(digits['data'].shape[1])])
pd_digits = pd.concat([pd_digits,
                       pd.Series(digits.target,
                                 name = 'target')],
                        axis = 1)

oml_digits = oml.push(pd_digits)

# Sample 20% and 80% of the data.
splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
[len(split) for split in splits]

# Split the data into four sets.
splits = oml_digits.split(ratio = (.25, .25, .25, .25),
                          use_hash = False)
[len(split) for split in splits]

# Perform stratification on the target column.
splits = oml_digits.split(strata_cols=['target'])
[spl.shape for spl in splits]

# Verify that the stratified sampling generates splits in which
# all of the different categories of digits (digits 0~9)  
# are present in each split.
[spl['target'].drop_duplicates().sort_values().pull() for spl in splits]

# Hash on the target column.
splits = oml_digits.split(hash_cols=['target'])
[spl.shape for spl in splits]

# Verify that the different categories of digits (digits 0~9) are present
# in only one of the splits generated by hashing on the category column.
```
# Split the data randomly into 4 consecutive folds.
folds = oml_digits.KFold(n_splits=4)
[(len(fold[0]), len(fold[1])) for fold in folds]

### Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> digits = datasets.load_digits()
>>> pd_digits = pd.DataFrame(digits.data,
...                          columns=['IMG'+str(i) for i in
...                          range(digits['data'].shape[1])])
>>> pd_digits = pd.concat([pd_digits,
...                        pd.Series(digits.target,
...                                   name = 'target')],
...                        axis = 1)
>>> oml_digits = oml.push(pd_digits)

>>> # Sample 20% and 80% of the data.
... splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
... [len(split) for split in splits]
[351, 1446]

>>> # Split the data into four sets.
... splits = oml_digits.split(ratio = (.25, .25, .25, .25),
...                           use_hash = False)
... [len(split) for split in splits]
[432, 460, 451, 454]

>>> # Perform stratification on the target column.
... splits = oml_digits.split(strata_cols=['target'])
... [split.shape for split in splits]
[(1285, 65), (512, 65)]

>>> # Verify that the stratified sampling generates splits in which
... # all of the different categories of digits (digits 0-9)
... # are present in each split.
... [split['target'].drop_duplicates().sort_values().pull()
...  for split in splits]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]

>>> # Hash on the target column
... splits = oml_digits.split(hash_cols=['target'])
... [split.shape for split in splits]
[(899, 65), (898, 65)]

>>> # Verify that the different categories of digits (digits 0-9) are
... # present
... # in only one of the splits generated by hashing on the category
```
Explore Data

OML4Py provides methods that enable you to perform exploratory data analysis and common statistical operations.

These methods are described in the following topics.

About the Exploratory Data Analysis Methods

OML4Py provides methods that enable you to perform exploratory data analysis.

The following table lists methods of OML4Py data type classes with which you can perform common statistical operations and indicates whether the class supports the method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>oml.Boolea n</th>
<th>oml.Byte s</th>
<th>oml.Floa t</th>
<th>oml.Strin g</th>
<th>oml.DataFram e</th>
</tr>
</thead>
<tbody>
<tr>
<td>corr</td>
<td>Computes pairwise correlation between all columns in an oml.DataFrame where possible, given the type of coefficient.</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>count</td>
<td>Computes the number of elements that are not NULL in the series data object or in each column of an oml.DataFrame.</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>crosstab</td>
<td>Computes a cross-tabulation of two or more columns in an oml.DataFrame.</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>cumsum</td>
<td>Computes the cumulative sum after an oml.Float series data object is sorted, or of each float or Boolean column after an oml.DataFrame object is sorted.</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>describe</td>
<td>Computes descriptive statistics that summarize the central tendency, dispersion, and shape of an oml series data distribution, or of each column in an oml.DataFrame.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>kurtosis</td>
<td>Computes the kurtosis of the values in an oml.Float series data object, or for each float column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>max</td>
<td>Returns the maximum value in a series data object or in each column in an oml.DataFrame.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>mean</td>
<td>Computes the mean of the values in an oml.Float series object, or for each float or Boolean column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>median</td>
<td>Computes the median of the values in an oml.Float series object, or for each float column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>min</td>
<td>Returns the minimum value in a series data object or of each column in an oml.DataFrame.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>nunique</td>
<td>Computes the number of unique values in a series data object or in each column of an oml.DataFrame.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>pivot_table</td>
<td>Converts an oml.DataFrame to a spreadsheet-style pivot table.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>sort_values</td>
<td>Sorts the values in a series data object or sorts the rows in an oml.DataFrame.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Table 4-2  (Cont.) Data Exploration Methods Supported by Data Type Classes

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>oml.Boolea n</th>
<th>oml.Byte s</th>
<th>oml.Floa t</th>
<th>oml.Strin g</th>
<th>oml.DataFram e</th>
</tr>
</thead>
<tbody>
<tr>
<td>skew</td>
<td>Computes the skewness of the values in an oml.Float data series object or of each float column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>std</td>
<td>Computes the standard deviation of the values in an oml.Float data series object or in each float or Boolean column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>sum</td>
<td>Computes the sum of the values in an oml.Float data series object or of each float or Boolean column in an oml.DataFrame.</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Correlate Data

Use the `corr` method to perform Pearson, Spearman, or Kendall correlation analysis across columns where possible in an oml.DataFrame object.

For details about the function arguments, invoke `help(oml.DataFrame.corr)` or see Oracle Machine Learning for Python API Reference.

Example 4-9  Performing Basic Correlation Calculations

This example first creates a temporary database table, with its corresponding proxy oml.DataFrame object `oml_df1`, from the pandas.DataFrame object `df`. It then verifies the correlation computed between columns A and B, which gives 1, as expected. The values in B are twice the values in A element-wise. The example also changes a value field in `df` and creates a NaN entry. It then creates a temporary database table, with the corresponding proxy oml.DataFrame object `oml_df2`. Finally, it invokes the `corr` method on `oml_df2` with `skipna` set to `True` (the default) and then `False` to compare the results.

```python
import oml
import pandas as pd

df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)]})
oml_df1 = oml.push(df)

# Verify that the correlation between column A and B is 1.
oml_df1.corr()

# Change a value to test the change in the computed correlation result.
df.loc[2, 'A'] = 1.5
```
# Change an entry to NaN (not a number) to test the 'skipna'
# parameter in the corr method.
df.loc[1, 'B'] = None

# Push df to the database using the floating point column type
# because NaNs cannot be used in Oracle numbers.
oml_df2 = oml.push(df, oranumber=False)

# By default, 'skipna' is True.
oml_df2.corr()
oml_df2.corr(skipna=False)

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)])})
>>> oml_df1 = oml.push(df)

# Verify that the correlation between column A and B is 1.
... oml_df1.corr()
    A   B
A  1.0  1.0
B  1.0  1.0

# Change a value to test the change in the computed correlation result.
... df.loc[2, 'A'] = 1.5

# Change an entry to NaN (not a number) so to test the 'skipna'
... # parameter in the corr method.
... df.loc[1, 'B'] = None

# Push df to the database using the floating point column type
... # because NaNs cannot be used in Oracle numbers.
... oml_df2 = oml.push(df, oranumber=False)

# By default, 'skipna' is True.
... oml_df2.corr()
    A   B
A  1.000000  0.981981
B  0.981981  1.000000

>>> oml_df2.corr(skipna=False)
    A   B
A  1.0   NaN
B  NaN  1.0
```
Cross-Tabulate Data

Use the crosstab method to perform cross-column analysis of an oml.DataFrame object and the pivot_table method to convert an oml.DataFrame to a spreadsheet-style pivot table.

Cross-tabulation is a statistical technique that finds an interdependent relationship between two columns of values. The crosstab method computes a cross-tabulation of two or more columns. By default, it computes a frequency table for the columns unless a column and an aggregation function have been passed to it.

The pivot_table method converts a data set into a pivot table. Due to the database 1000 column limit, pivot tables with more than 1000 columns are automatically truncated to display the categories with the most entries for each column value.

For details about the method arguments, invoke help(oml.DataFrame.crosstab) or help(oml.DataFrame.pivot_table), or see Oracle Machine Learning for Python API Reference.

Example 4-10  Producing Cross-Tabulation and Pivot Tables

This example demonstrates the use of the crosstab and pivot_table methods.

```python
import pandas as pd
import oml

x = pd.DataFrame({
    'GENDER': ['M', 'M', 'F', 'M', 'F', 'F', None, 'F', 'M', 'F', 'M', 'F'],
    'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1,
               39.6, 46.4, 12, 25.3, 37.5],
    'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93,
                 .89, .84, .91, .95]
})

x = oml.push(x)

# Find the categories that the most entries belonged to.
x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)

# For each gender value and across all entries, find the ratio of entries
# with different hand values.
x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)

# Find the mean speed across all gender and hand combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED')

# Find the median accuracy and speed for every gender and hand combination.
x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)

# Find the max and min speeds for every gender and hand combination and
# across all combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED',
```

Chapter 4
Explore Data
Listing for This Example

```python
>>> import pandas as pd
>>> import oml

>>> x = pd.DataFrame({
...     'GENDER': ['M', 'M', 'F', 'M', 'F', 'F', None, 'F', 'M', 'F', 'F', None],
...     'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R', 'R', 'R', 'R', 'R'],
...     'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1, 39.6, 46.4, 12, 25.3, 37.5],
...     'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93, .89, .84, .91, .95]
... })
>>> x = oml.push(x)

>>> # Find the categories that the most entries belonged to.
... x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)

GENDER  HAND  count
0      F     R      5
1      M     L      2
2      M     R      2
3      M  None      1
4      F     L      1
5   None     R      1

>>> # For each gender value and across all entries, find the ratio of entries
# with different hand values.
... x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)

GENDER  count_(L)  count_(R)  count_(None)
0   None   0.000000   1.000000      0.000000
1      F   0.166667   0.833333      0.000000
2      M   0.400000   0.400000      0.200000
3    All   0.250000   0.666667      0.083333

>>> # Find the mean speed across all gender and hand combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED')

GENDER  mean(SPEED)_(L)  mean(SPEED)_(R)  mean(SPEED)_(None)
0   None              NaN            46.40                 NaN
1      F            34.10            40.78                 NaN
2      M            45.85            27.85                29.3

>>> # Find the median accuracy and speed for every gender and hand combination.
... x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)

GENDER  median(ACCURACY)_(L)  median(ACCURACY)_(R)  median(ACCURACY)_(None)
0   None                   NaN                   NaN                   NaN
```
0.890                      NaN
1      F                  0.96                 0.870
NaN
2      M                  0.91                 0.925
0.97

median(SPEED)_(L)  median(SPEED)_(R)  median(SPEED)_(None)
0                NaN              46.40                   NaN
1              34.10              39.60                   NaN
2              45.85              27.85                  29.3
>>>
>>> # Find the max and min speeds for every gender and hand combination and
... # across all combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED',
...                aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
...                margins = True)
GENDER  max(SPEED)_(L)  max(SPEED)_(R)  max(SPEED)_(None)
max(SPEED)_(All)  \
0   None             NaN            46.4                NaN
46.4
1      F            34.1            60.8                NaN
60.8
2      M            51.2            30.4               29.3
51.2
3    All            51.2            60.8               29.3
60.8

min(SPEED)_(L)  min(SPEED)_(R)  min(SPEED)_(None)  min(SPEED)_(All)
0             NaN            46.4                NaN              46.4
1            34.1            12.0                NaN              12.0
2            40.5            25.3               29.3              25.3
3            34.1            12.0               29.3              12.0

Mutate Data

In preparing data for analysis, a typical operation is to mutate data by reformatting it or
deriving new columns and adding them to the data set.

These examples demonstrate methods of formatting data and deriving columns.

import pandas as pd
import oml

# Create a shopping cart data set.
shopping_cart = pd.DataFrame(
    {'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
                   'pork_loin', 'whole_milk', 'egg_custard'],
     'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
                   'dairy', 'bakery'],
     'Quantity': [1, 2.6, 4, 1, 1.9, 1, 1],
     'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
)}
oml_cart = oml.push(shopping_cart)
oml_cart
# Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
# rounded to 2 decimal places.
price = oml_cart['Quantity']*(oml_cart['Unit_price'])
type(price)
price
oml_cart = oml_cart.concat({"Price": price.round(2)})

# Count the pattern 'egg' in the 'Item_name' column.
egg_pattern = oml_cart['Item_name'].count_pattern('egg')
type(egg_pattern)
oml_cart.concat({"Egg_pattern": egg_pattern})

# Find the start index of substring 'pork' in the 'Item_name' column.
pork_startInd = oml_cart['Item_name'].find('pork')
type(pork_startInd)
oml_cart.concat({"Pork_startInd": pork_startInd})

# Check whether items are of grocery category.
is_grocery=oml_cart['Item_type']=='grocery'
type(is_grocery)
oml_cart.concat({"Is_grocery": is_grocery})

# Calculate the length of item names.
name_length=oml_cart['Item_name'].len()
type(name_length)
oml_cart.concat({"Name_length": name_length})

# Get the ceiling, floor, exponential, logarithm and square root
# of the 'Price' column.
oml_cart['Price'].ceil()
oml_cart['Price'].floor()
oml_cart['Price'].exp()
oml_cart['Price'].log()
oml_cart['Price'].sqrt()

Listing for This Example

>>> import pandas as pd
>>> import oml

>>> # Create a shopping cart data set.
... shopping_cart = pd.DataFrame({
...   'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
...                  'pork_loin', 'whole_milk', 'egg_custard'],
...   'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
...                  'dairy', 'bakery'],
...   'Quantity': [1, 2.6, 4, 1.9, 1, 1],
...   'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
... })

>>> oml_cart = oml.push(shopping_cart)

>>> oml_cart
   Item_name  Item_type  Quantity  Unit_price
0     paper_towel  grocery       1.0       1.19
1 ground_pork  meat  2.6  2.79
2 tofu  grocery  4.0  0.99
3 eggs  dairy  1.0  2.49
4 pork_loin  meat  1.9  3.19
5 whole_milk  dairy  1.0  2.50
6 egg_custard  bakery  1.0  3.99

>>> # Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
... # rounded to 2 decimal places.
... price = oml_cart['Quantity']*(oml_cart['Unit_price'])
>>> type(price)
<class 'oml.core.float.Float'>
>>> price
[1.19, 7.254, 3.96, 2.49, 6.061, 2.5, 3.99]
>>> oml_cart = oml_cart.concat({'Price': price.round(2)})

>>> # Count the pattern 'egg' in the 'Item_name' column.
... egg_pattern = oml_cart['Item_name'].count_pattern('egg')
>>> type(egg_pattern)
<class 'oml.core.float.Float'>

>>> oml_cart.concat({'Egg_pattern': egg_pattern})

<table>
<thead>
<tr>
<th>Item_name</th>
<th>Item_type</th>
<th>Quantity</th>
<th>Unit_price</th>
<th>Price</th>
<th>Egg_pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper_towel</td>
<td>grocery</td>
<td>1.0</td>
<td>1.19</td>
<td>1.19</td>
<td>0</td>
</tr>
<tr>
<td>ground_pork</td>
<td>meat</td>
<td>2.6</td>
<td>2.79</td>
<td>7.25</td>
<td>0</td>
</tr>
<tr>
<td>tofu</td>
<td>grocery</td>
<td>4.0</td>
<td>0.99</td>
<td>3.96</td>
<td>0</td>
</tr>
<tr>
<td>eggs</td>
<td>dairy</td>
<td>1.0</td>
<td>2.49</td>
<td>2.49</td>
<td>1</td>
</tr>
<tr>
<td>pork_loin</td>
<td>meat</td>
<td>1.9</td>
<td>3.19</td>
<td>6.06</td>
<td>0</td>
</tr>
<tr>
<td>whole_milk</td>
<td>dairy</td>
<td>1.0</td>
<td>2.50</td>
<td>2.50</td>
<td>0</td>
</tr>
<tr>
<td>egg_custard</td>
<td>bakery</td>
<td>1.0</td>
<td>3.99</td>
<td>3.99</td>
<td>1</td>
</tr>
</tbody>
</table>

>>> # Find the start index of substring 'pork' in the 'Item_name' column.
... pork_startInd = oml_cart['Item_name'].find('pork')
>>> type(pork_startInd)
<class 'oml.core.float.Float'>

>>> oml_cart.concat({'Pork_startInd': pork_startInd})

<table>
<thead>
<tr>
<th>Item_name</th>
<th>Item_type</th>
<th>Quantity</th>
<th>Unit_price</th>
<th>Price</th>
<th>Pork_startInd</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper_towel</td>
<td>grocery</td>
<td>1.0</td>
<td>1.19</td>
<td>1.19</td>
<td>-1</td>
</tr>
<tr>
<td>ground_pork</td>
<td>meat</td>
<td>2.6</td>
<td>2.79</td>
<td>7.25</td>
<td>7</td>
</tr>
<tr>
<td>tofu</td>
<td>grocery</td>
<td>4.0</td>
<td>0.99</td>
<td>3.96</td>
<td>-1</td>
</tr>
<tr>
<td>eggs</td>
<td>dairy</td>
<td>1.0</td>
<td>2.49</td>
<td>2.49</td>
<td>-1</td>
</tr>
<tr>
<td>pork_loin</td>
<td>meat</td>
<td>1.9</td>
<td>3.19</td>
<td>6.06</td>
<td>0</td>
</tr>
<tr>
<td>whole_milk</td>
<td>dairy</td>
<td>1.0</td>
<td>2.50</td>
<td>2.50</td>
<td>-1</td>
</tr>
<tr>
<td>egg_custard</td>
<td>bakery</td>
<td>1.0</td>
<td>3.99</td>
<td>3.99</td>
<td>-1</td>
</tr>
</tbody>
</table>

>>> # Check whether items are of grocery category.
... is_grocery=oml_cart['Item_type']=='grocery'

>>> type(is_grocery)
<class 'oml.core.boolean.Boolean'>

>>> oml_cart.concat({'Is_grocery': is_grocery})

<table>
<thead>
<tr>
<th>Item_name</th>
<th>Item_type</th>
<th>Quantity</th>
<th>Unit_price</th>
<th>Price</th>
<th>Is_grocery</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper_towel</td>
<td>grocery</td>
<td>1.0</td>
<td>1.19</td>
<td>1.19</td>
<td>True</td>
</tr>
<tr>
<td>ground_pork</td>
<td>meat</td>
<td>2.6</td>
<td>2.79</td>
<td>7.25</td>
<td>False</td>
</tr>
<tr>
<td>tofu</td>
<td>grocery</td>
<td>4.0</td>
<td>0.99</td>
<td>3.96</td>
<td>True</td>
</tr>
<tr>
<td>eggs</td>
<td>dairy</td>
<td>1.0</td>
<td>2.49</td>
<td>2.49</td>
<td>False</td>
</tr>
<tr>
<td>pork_loin</td>
<td>meat</td>
<td>1.9</td>
<td>3.19</td>
<td>6.06</td>
<td>False</td>
</tr>
</tbody>
</table>
5  whole_milk  dairy  1.0  2.50  2.50  False
6  egg_custard bakery  1.0  3.99  3.99  False

>>> # Calculate the length of item names.
... name_length=oml_cart['Item_name'].len()
>>> type(name_length)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Name_length': name_length})

<table>
<thead>
<tr>
<th>Item_name</th>
<th>Item_type</th>
<th>Quantity</th>
<th>Unit_price</th>
<th>Price</th>
<th>Name_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper_towel</td>
<td>grocery</td>
<td>1.0</td>
<td>1.19</td>
<td>1.19</td>
<td>11</td>
</tr>
<tr>
<td>ground_pork</td>
<td>meat</td>
<td>2.6</td>
<td>2.79</td>
<td>7.25</td>
<td>11</td>
</tr>
<tr>
<td>tofu</td>
<td>grocery</td>
<td>4.0</td>
<td>0.99</td>
<td>3.96</td>
<td>4</td>
</tr>
<tr>
<td>eggs</td>
<td>dairy</td>
<td>1.0</td>
<td>2.49</td>
<td>2.49</td>
<td>4</td>
</tr>
<tr>
<td>pork_loin</td>
<td>meat</td>
<td>1.9</td>
<td>3.19</td>
<td>6.06</td>
<td>9</td>
</tr>
<tr>
<td>whole_milk</td>
<td>dairy</td>
<td>1.0</td>
<td>2.50</td>
<td>2.50</td>
<td>10</td>
</tr>
<tr>
<td>egg_custard</td>
<td>bakery</td>
<td>1.0</td>
<td>3.99</td>
<td>3.99</td>
<td>11</td>
</tr>
</tbody>
</table>

>>> # Get the ceiling, floor, exponential, logarithm and square root...# of the 'Price' column.
... oml_cart['Price'].ceil()
[2, 8, 4, 3, 7, 3, 4]
>>> oml_cart['Price'].floor()
[1, 7, 3, 2, 6, 2, 3]
>>> oml_cart['Price'].exp()
[3.2870812073831184, 1408.1048482046956, 52.45732594909905, 12.061276120444719, 428.37543685928694, 12.182493960703473, 54.05488936332659]
>>> oml_cart['Price'].log()
[0.173953307123438, 1.9810014688665833, 1.3762440252663892, 0.9122827104766162, 1.801709800081223, 0.9162907318741551, 1.3837912309017721]
>>> oml_cart['Price'].sqrt()
[1.0908712114635715, 2.692582403567252, 1.98997487421324, 1.57797338380595, 2.4617067250182343, 1.5811388300841898, 1.99749845543818]

Sort Data

The sort_values function enables flexible sorting of an oml.DataFrame along one or more columns specified by the by argument, and returns an oml.DataFrame.

Example 4-11  Sorting Data

The following example demonstrate these operations.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
```
y = pd.DataFrame(list(map(lambda x:
    {0: 'setosa', 1: 'versicolor',
     2: 'virginica'}[x], iris.target)),
        columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Modify the data set by replacing a few entries with NaNs to test
# how the na_position parameter works in the sort_values method.
Iris = oml_iris.pull()
Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
Iris['Petal_Length'].replace({1.5: None}, inplace=True)
Iris['Petal_Width'].replace({2.3: None}, inplace=True)

# Create another table using the changed data.
oml_iris2 = oml.create(Iris, table = 'IRIS2')

# Sort the data set first by Sepal_Length then by Sepal_Width
# in descending order and display the first 5 rows of the
# sorted result.
oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
                      ascending=False).head()

# Display the last 5 rows of the data set.
oml_iris2.tail()

# Sort the last 5 rows of the iris data set first by Petal_Length
# then by Petal_Width. By default, rows with NaNs are placed
# after the other rows when the sort keys are the same.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])

# Sort the last 5 rows of the iris data set first by Petal_Length
# and then by Petal_Width. When the values in these two columns
# are the same, place the row with a NaN before the other row.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
                            na_position = 'first')

oml.drop('IRIS')
oml.drop('IRIS2')

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

... # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
... x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
... y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...        columns = ['Species'])
... # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

... # Modify the data set by replacing a few entries with NaNs to test
... # how the na_position parameter works in the sort_values method.
... Iris = oml_iris.pull()
... Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
... Iris['Petal_Length'].replace({1.5: None}, inplace=True)
... Iris['Petal_Width'].replace({2.3: None}, inplace=True)

... # Create another table using the changed data.
... oml_iris2 = oml.create(Iris, table = 'IRIS2')

... # Sort the data set first by Sepal_Length then by Sepal_Width
... # in descending order and display the first 5 rows of the
... # sorted result.
... oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
...                        ascending=False).head()

... # Display the last 5 rows of the data set.
... oml_iris2.tail()

... # Sort the last 5 rows of the iris data set first by Petal_Length
... # then by Petal_Width. By default, rows with NaNs are placed
... # after the other rows when the sort keys are the same.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])

... # Sort the last 5 rows of the iris data set first by Petal_Length
... # and then by Petal_Width. When the values in these two columns
... # are the same, place the row with a NaN before the other row.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
...                             na_position = 'first')

oml.drop('IRIS')
oml.drop('IRIS2')
... 2: 'virginica')[x], iris.target)),
...
columns = ['Species'])

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Modify the data set by replacing a few entries with NaNs to test
... # how the na_position parameter works in the sort_values method.
... Iris = oml_iris.pull()
>>> Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
>>> Iris['Petal_Length'].replace({1.5: None}, inplace=True)
>>> Iris['Petal_Width'].replace({2.3: None}, inplace=True)

>>> # Create another table using the changed data.
... oml_iris2 = oml.create(Iris, table = 'IRIS2')

>>> # Sort the data set first by 'Sepal_Length' then by 'Sepal_Width'
... # in descending order and displays the first 5 rows of the
... # sorted result.
... oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
...                        ascending=False).head()

    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           7.9          3.8           6.4          2.0  virginica
1           7.7          3.8           6.7          2.2  virginica
2           7.7          3.0           6.1          NaN  virginica
3           7.7          2.8           6.7          2.0  virginica
4           7.7          2.6           6.9          NaN  virginica

>>> # Display the last 5 rows of the data set.
... oml_iris2.tail()

    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.7          3.0           5.2          NaN  virginica
1           6.3          2.5           5.0          1.9  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.2          3.4           5.4          NaN  virginica
4           5.9          3.0           5.1          1.8  virginica

>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # then by 'Petal_Width'. By default, rows with NaNs are placed
... # after the other rows when the sort keys are the same.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])

    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.3          2.5           5.0          1.9  virginica
1           5.9          3.0           5.1          1.8  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.2          3.4           5.4          NaN  virginica
4           6.2          3.4           5.4          NaN  virginica

>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # and then by 'Petal_Width'. When the values in these two columns
... # are the same, place the row with a NaN before the other row.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
...                              na_position = 'first')

    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.3          2.5           5.0          1.9  virginica
Summarize Data

The `describe` method calculates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column.

You can also specify the types of columns to include or exclude from the results.

With the `sum` and `cumsum` methods, you can compute the sum and cumulative sum of each Float or Boolean column of an `oml.DataFrame`.

The `describe` method supports finding the following statistics:

- Mean, minimum, maximum, median, top character, standard deviation
- Number of not-Null values, unique values, top characters
- Percentiles between 0 and 1

**Example 4-12  Calculating Descriptive Statistics**

The following example demonstrates these operations.

```python
import pandas as pd
import oml

df = pd.DataFrame({
    'numeric': [1, 1.4, -4, 3.145, 5, None],
    'string': [None, None, 'a', 'a', 'a', 'b'],
    'bytes': [b'a', b'b', b'c', b'c', b'd', b'e']})

oml_df = oml.push(df, dbtypes = {
    'numeric': 'BINARY_DOUBLE',
    'string': 'CHAR(1)',
    'bytes': 'RAW(1)'})

# Combine a Boolean column with oml_df.
oml_bool = oml_df['numeric'] > 3
oml_df = oml_df.concat(oml_bool)
oml_df.rename({'COL4': 'boolean'})

# Describe all of the columns.
oml_df.describe(include='all')

# Exclude Float columns.
oml_df.describe(exclude=[oml.Float])

# Get the sum of values in each Float or Boolean column.
oml_df.sum()

# Find the cumulative sum of values in each Float or Boolean column
# after oml_df is sorted by the bytes column in descending order.
```
oml_df.cumsum(by = 'bytes', ascending = False)

# Compute the skewness of values in the Float columns.
oml_df.skew()

# Find the median value of Float columns.
oml_df.median()

# Calculate the kurtosis of Float columns.
oml_df.kurtosis()

Listing for This Example

>>> import pandas as pd
>>> import oml

>>> df = pd.DataFrame({
    'numeric': [1, 1.4, -4, 3.145, 5, None],
    'string': [None, None, 'a', 'a', 'a', 'b'],
    'bytes': [b'a', b'b', b'c', b'c', b'd', b'e']})

>>> oml_df = oml.push(df, dbtypes = {'numeric': oml.BINARY_DOUBLE,
    'string': oml.CHAR(1),
    'bytes': oml.RAW(1)})

>>> # Combine a Boolean column with oml_df.
... oml_bool = oml_df['numeric'] > 3
>>> oml_df = oml_df.concat(oml_bool)
>>> oml_df.rename({'COL4':'boolean'})

   bytes  numeric  string  boolean
0   b'a'    1.000   None    False
1   b'b'    1.400   None    False
2   b'c'   -4.000     a    False
3   b'c'    3.145     a     True
4   b'd'    5.000     a     True
5   b'e'      NaN    b     True

>>> # Describe all of the columns.
... oml_df.describe(include='all')

   bytes  numeric  string  boolean
count      6  5.000000      4       6
unique     5     NaN      2       2
top   b'c'   NaN      a    False
freq       2     NaN      3       3
mean     NaN  1.309000   NaN    NaN
std     NaN  3.364655   NaN    NaN
min     NaN  -4.000000  NaN    NaN
25%      NaN  1.000000  NaN    NaN
50%      NaN  1.400000  NaN    NaN
75%      NaN  3.145000  NaN    NaN
max      NaN  5.000000  NaN    NaN

>>> # Exclude Float columns.
... oml_df.describe(exclude=[oml.Float])

   bytes  string  boolean
count     6      4       6
unique    5     NaN      2
top   b'c'   NaN      a    False
freq      2     NaN      3
mean  NaN  1.309000   NaN    NaN
std  NaN  3.364655   NaN    NaN
min  NaN  -4.000000  NaN    NaN
25%  NaN  1.000000  NaN    NaN
50%  NaN  1.400000  NaN    NaN
75%  NaN  3.145000  NaN    NaN
max  NaN  5.000000  NaN    NaN
Render Graphics

OML4Py provides functions for rendering graphical displays of data.

The `oml.boxplot` and `oml.hist` functions compute the statistics necessary to generate box and whisker plots or histograms in-database for scalability and performance.

OML4Py uses the `matplotlib` library to render the output. You can use methods of `matplotlib.pyplot` to customize the created images and `matplotlib.pyplot.show` to show the images. By default, rendered graphics have the same properties as those stored in `matplotlib.rcParams`.

For the parameters of the `oml.boxplot` and `oml.hist` functions, invoke `help(oml.boxplot)` or `help(oml.hist)`, or see Oracle Machine Learning for Python API Reference.
Generate a Box Plot

Use the `oml.boxplot` function to generate a box and whisker plot for every column of \( x \) or for every column object in \( x \).

**Example 4-13 Using the oml.boxplot Function**

This example first loads the wine data set from `sklearn` and creates the `pandas.DataFrame` object `wine_data`. It then creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_wine`, from `wine_data`. It draws a box and whisker plot on every column with the index ranging from 8 to 12 (not including 12) in `oml_wine`. The arguments `showmeans` and `meanline` are set to `True` to show the arithmetic means and to render the mean as a line spanning the full width of the box. The argument `patch_artist` is set to `True` to have the boxes drawn with Patch artists.

```python
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

wine = datasets.load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)

oml.graphics.boxplot(oml_wine[:,8:12], showmeans=True,
                      meanline=True, patch_artist=True,
                      labels=oml_wine.columns[8:12])
plt.title('Distribution of Wine Attributes')
plt.show()
```

The output of the example is the following.

![Box plot for wine data set](image)

The image shows a box and whisker plot for each of the four columns of the wine data set: Proanthocyanins, Color intensity, Hue, and OD280/OD315 of diluted wines. The boxes extend from the lower to upper quartile values of the data, with a solid orange
line at the median. The whiskers that extend from the box show the range of the data. The caps are the horizontal lines at the ends of the whiskers. Flier or outlier points are those past the ends of the whiskers. The mean is shown as a green dotted line spanning the width of the each box.

**Generate a Histogram**

Use the oml.hist function to compute and draw a histogram for every data set column contained in x.

**Example 4-14 Using the oml.hist Function**

This example first loads the wine data set from sklearn and creates the pandas.DataFrame object wine_data. It then creates a temporary database table, with its corresponding proxy oml.DataFrame object oml_wine, from wine_data. Next it draws a histogram on the proline column of oml_wine. The argument bins specifies generating ten equal-width bins. Argument color specifies filling the bars with the color purple. Arguments linestyle and edgecolor are set to draw the bar edges as solid lines in pink.

```python
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine

wine = load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)

oml.graphics.hist(oml_wine['proline'], bins=10, color='red', linestyle='solid', edgecolor='white')

plt.title('Proline content in Wine')
plt.xlabel('proline content')
plt.ylabel('# of wine instances')
plt.show()
```

The output of the example is the following.

![Proline content in Wine](image)

The image shows a traditional bar-type histogram for the Proline column of the wine data set. The range of proline values is divided into 10 bins of equal size. The height of the rectangular
bar for each bin indicates the number of wine instances in each bin. The bars are red with solid white edges.
OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

OML4Py has classes that provide access to in-database Oracle Machine Learning algorithms.

These classes are described in the following topics.

- About Machine Learning Classes and Algorithms
- About Model Settings
- Shared Settings
- Export Oracle Machine Learning for Python Models
- Model Explainability
- Attribute Importance
- Association Rules
- Decision Tree
- Expectation Maximization
- Explicit Semantic Analysis
- Generalized Linear Model
- k-Means
- Naive Bayes
- Neural Network
- Random Forest
- Singular Value Decomposition
- Support Vector Machine

About Machine Learning Classes and Algorithms

These classes provide access to in-database machine learning algorithms.

Algorithm Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Algorithm</th>
<th>Function of Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.ai</td>
<td>Minimum Description Length</td>
<td>Attribute importance for classification or regression</td>
<td>Ranks attributes according to their importance in predicting a target.</td>
</tr>
<tr>
<td>Class</td>
<td>Algorithm</td>
<td>Function of Algorithm</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>oml.ar</td>
<td>Apriori</td>
<td>Association rules</td>
<td>Performs market basket analysis by identifying co-occurring items (frequent itemsets) within a set.</td>
</tr>
<tr>
<td>oml.dt</td>
<td>Decision Tree</td>
<td>Classification</td>
<td>Extracts 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.</td>
</tr>
<tr>
<td>oml.em</td>
<td>Expectation Maximization</td>
<td>Clustering</td>
<td>Performs probabilistic clustering based on a density estimation algorithm.</td>
</tr>
<tr>
<td>oml.esa</td>
<td>Explicit Semantic Analysis</td>
<td>Feature extraction</td>
<td>Extracts text-based features from a corpus of documents. Performs document similarity comparisons.</td>
</tr>
<tr>
<td>oml.glm</td>
<td>Generalized Linear Model</td>
<td>Classification Regression</td>
<td>Implements logistic regression for classification of binary targets and linear regression for continuous targets.</td>
</tr>
<tr>
<td>oml.km</td>
<td>k-Means</td>
<td>Clustering</td>
<td>Uses unsupervised learning to group data based on similarity into a predetermined number of clusters.</td>
</tr>
<tr>
<td>oml.nb</td>
<td>Naive Bayes</td>
<td>Classification</td>
<td>Makes predictions by deriving the probability of a prediction from the underlying evidence, as observed in the data.</td>
</tr>
<tr>
<td>oml.nn</td>
<td>Neural Network</td>
<td>Classification Regression</td>
<td>Learns from examples and tunes the weights of the connections among the neurons during the learning process.</td>
</tr>
<tr>
<td>oml.rf</td>
<td>Random Forest</td>
<td>Classification</td>
<td>Provides an ensemble learning technique for classification of data.</td>
</tr>
<tr>
<td>oml.svd</td>
<td>Singular Value Decomposition</td>
<td>Feature extraction</td>
<td>Performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices.</td>
</tr>
<tr>
<td>oml.svm</td>
<td>Support Vector Machine</td>
<td>Anomaly detection Classification Regression</td>
<td>Builds a model that is a profile of a class, which, when the model is applied, identifies cases that are somehow different from that profile.</td>
</tr>
</tbody>
</table>

**Repeatability Results**

You can use the `case_id` parameter in the `fit` method of the OML4Py machine learning algorithm classes to achieve repeatable sampling, data splits (train and held aside), and random data shuffling.

**Persisting Models**

In-database models created through the OML4Py API exist as temporary objects that are dropped when the database connection ends unless you take one of the following actions:
• Save a default-named model object in a datastore, as in the following example:

```python
regr2 = oml.glm("regression")
oml.ds.save(regr2, 'regression2')
```

• Use the `model_name` parameter in the `fit` function when building the model, as in the following example:

```python
regr2 = regr2.fit(X, y, model_name = 'regression2')
```

• Change the name of an existing model using the `model_name` function of the model, as in the following example:

```python
regr2(model_name = 'myRegression2')
```

To drop a persistent named model, use the `oml.drop` function.

Creating a Model from an Existing In-Database Model

You can create an OML4Py model as a proxy object for an existing in-database machine learning model. The in-database model could have been created through OML4Py, OML4SQL, or OML4R. To do so, when creating the OML4Py, specify the name of the existing model and, optionally, the name of the owner of the model, as in the following example.

```python
ar_mod = oml.ar(model_name = 'existing_ar_model', model_owner = 'SH', **setting)
```

An OML4Py model created this way persists until you drop it with the `oml.drop` function.

Scoring New Data with a Model

For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.

For in-database models, you can use the SQL `PREDICTION` function on model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

Deploying Models Through a REST API

The REST API for Oracle Machine Learning Services provides REST endpoints hosted on an Oracle Autonomous Database instance. These endpoints allow you to store OML models along with their metadata, and to create scoring endpoints for the models.

About Model Settings

You can specify settings that affect the characteristics of a model.

Some settings are general, some are specific to an Oracle 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 specify the settings with the `**params` parameter when instantiating the model or later by using the `set_params` method of the model.

For the `__init__` method, the argument can be key-value pairs or a dict. Each list element’s name and value refer to a machine learning algorithm parameter setting name and value, respectively. The setting value must be numeric or a string.

The argument for the `**params` parameter of the `set_params` method is a dict object mapping a str to a str. The key should be the name of the setting, and the value should be the new setting.

**Example 5-1  Specifying Model Settings**

This example shows the creation of an Expectation Maximization (EM) model and the changing of a setting. For the complete code of the EM model example, see Example 5-10.

```python
# Specify settings.
setting = {'emcs_num_iterations': 100}
# Create an EM model object
em_mod = em(n_clusters = 2, **setting)

# Intervening code not shown.

# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
```

## Shared Settings

These settings are common to all of the OML4Py machine learning classes.

The following table lists the settings that are shared by all OML4Py models.

**Table 5-1  Shared Model Settings**

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMS_DETAILS</td>
<td>• ODMS_ENABLE</td>
<td>Reduces the space that is used while creating a model, especially a partitioned model. The default value is ODMS_ENABLE.</td>
</tr>
<tr>
<td>ODMS_DISABLE</td>
<td>• ODMS_DISABLE</td>
<td>If the setting value is ODMS_ENABLE, then model tables and views are created along with the model. You can query the model with SQL.</td>
</tr>
<tr>
<td>ODMS_MAX_PARTITIONS</td>
<td>1 &lt; value &lt;= 1000000</td>
<td>The maximum number of partitions allowed for the model. The default is 1000.</td>
</tr>
</tbody>
</table>
### Table 5-1  (Cont.) Shared Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td><a href="#">ODMS_MISSING_VALUE_M EAN_MODE</a></td>
<td>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 is applicable to all algorithms.</td>
</tr>
<tr>
<td>ODMS_PARTITION_BUILD_TYPE</td>
<td><a href="#">ODMS_PARTITION_BUILD_INTR A</a></td>
<td>Controls the parallel building of partitioned models.</td>
</tr>
<tr>
<td></td>
<td><a href="#">ODMS_PARTITION_BUILD_INTER</a></td>
<td>• ODMS_PARTITION_BUILD_INTR_A builds each partition in parallel using all slaves.</td>
</tr>
<tr>
<td></td>
<td><a href="#">ODMS_PARTITION_BUILD_HYBRID</a></td>
<td>• ODMS_PARTITION_BUILD_INTER builds each partition entirely in a single slave, but multiple partitions may be built at the same time because multiple slaves are active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ODMS_PARTITION_BUILD_HYBRID combines the other two types and is recommended for most situations to adapt to dynamic environments. This is the default value.</td>
</tr>
<tr>
<td>ODMS_PARTITION_COLUMNS</td>
<td>Comma separated list of machine learning attributes</td>
<td>Requests the building of a partitioned model. The setting value is a comma-separated list of the machine learning attributes to be used to determine the in-list partition key values. These attributes are taken from the input columns, unless an XFORM_LIST parameter is passed to the model. If XFORM_LIST parameter is passed to the model, then the attributes are taken from the attributes produced by these transformations.</td>
</tr>
<tr>
<td>ODMS_TABLESPACE_NAME</td>
<td>tablespace_name</td>
<td>Specifies the tablespace in which to store the model. If you explicitly set 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 your default tablespace creates the resulting model content.</td>
</tr>
<tr>
<td>ODMS_SAMPLE_SIZE</td>
<td>0 &lt; value</td>
<td>Determines how many rows to sample (approximately). You can use this setting only if ODMS_SAMPLING is enabled. The default value is system determined.</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td><a href="#">ODMS_SAMPLING_ENABLE</a></td>
<td>Allows the user to request sampling of the build data. The default is ODMS_SAMPLING_DISABLE.</td>
</tr>
</tbody>
</table>

---

[ODMS_MISSING_VALUE_M EAN_MODE](#): replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time where appropriate.

[ODMS_PARTITION_BUILD_INTR A](#): builds each partition in parallel using all slaves.

[ODMS_PARTITION_BUILD_INTER](#): builds each partition entirely in a single slave, but multiple partitions may be built at the same time because multiple slaves are active.

[ODMS_PARTITION_BUILD_HYBRID](#): combines the other two types and is recommended for most situations to adapt to dynamic environments. This is the default value.
Table 5-1  (Cont.) Shared Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMS_TEXT_MAX_FEATURES</td>
<td>1 &lt;= value</td>
<td>The maximum number of distinct features, across all text attributes, to use from a document set passed to the model. The default is 3000. An oml.esa model has the default value of 300000.</td>
</tr>
<tr>
<td>ODMS_TEXT_MIN_DOCUMENTS</td>
<td>Non-negative value</td>
<td>This text processing setting controls how many documents a token needs to appear in to be used as a feature. The default is 1. An oml.esa model has the default value of 3.</td>
</tr>
<tr>
<td>ODMS_TEXT_POLICY_NAME</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Export Oracle Machine Learning for Python Models

You can export an oml model from Python and then score it in SQL.

Export a Model

With the export_sermodel function of an OML4Py algorithm model, you can export the model in a serialized format. You can then score that model in SQL. To save a model to a permanent table, you must pass in a name for the new table. If the model is partitioned, then you can optionally select an individual partition to export; otherwise all partitions are exported.

Note:

Any data transformations you apply to the data for model building you must also apply to the data for scoring with the imported model.

Example 5-2  Export a Trained oml.svm Model to a Database Table

This example creates the x and y variables using the iris data set. It then creates the persistent database table IRIS and the oml.DataFrame object oml_iris as a proxy for the table.

This example preprocesses the iris data set and splits the data set into training data and test data. It then fits an oml.svm model according to the training data of the data set, and saves the fitted model in a serialized format to a new table named svm_sermod in the database.

```python
import oml
import pandas as pd
from sklearn import datasets
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
```
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                            'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                           {0: 'setosa', 1: 'versicolor',
                            2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop('IRIS_TEST_DATA')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

df = oml.sync(table = "IRIS").pull()

# Add a case identifier column.
df.insert(0, 'ID', range(0,len(df)))

# Create training data and test data.
IRIS_TMP = oml.push(df).split()
train_x = IRIS_TMP[0].drop('Species')
train_y = IRIS_TMP[0]['Species']
test_dat = IRIS_TMP[1]

# Create the iris_test_data database table.
oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")

# Create an oml SVM model object.
svm_mod = oml.svm('classification',
                  svms_kernel_function =
                  'dbms_data_mining.svms_linear')

# Fit the SVM model with the training data.
svm_mod = svm_mod.fit(train_x, train_y, case_id = 'ID')

# Export the oml.svm model to a new table named 'svm_sermod'
# in the database.
svm_export = svm_mod.export_sermodel(table='svm_sermod')
type(svm_export)

# Show the first 10 characters of the BLOB content from the
# model export.
svm_export.pull()[0][1:10]

**Listing for This Example**

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
```
# Load the iris data set and create a pandas.DataFrame for it.
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                   columns = ['Sepal_Length', 'Sepal_Width',
...                              'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
...    oml.drop('IRIS_TEST_DATA')
...except:
...    pass

>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>> df = oml.sync(table = "IRIS").pull()

>>> # Add a case identifier column.
... df.insert(0, 'ID', range(0,len(df)))

>>> # Create training data and test data.
... IRIS_TMP = oml.push(df).split()
... train_x = IRIS_TMP[0].drop('Species')
... train_y = IRIS_TMP[0]['Species']
... test_dat = IRIS_TMP[1]

>>> # Create the iris_test_data database table.
... oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")

>>> # Create an oml SVM model object.
... svm_mod = oml.svm('classification',
...                   svms_kernel_function = 'dbms_data_mining.svms_linear')

>>> # Fit the SVM model with the training data.
... svm_mod = svm_mod.fit(train_x, train_y, case_id='ID')

>>> # Export the oml.svm model to a new table named 'svm_sermod'
... # in the database.
... svm_export = svm_mod.export_sermodel(table='svm_sermod')
... type(svm_export)
<class 'oml.core.bytes.Bytes'>

>>> # Show the first 10 characters of the BLOB content from the
... # model export.
... svm_export.pull()[0][1:10]
'b\xff\xfc\x00\x00\x02\x9c\x00\x00\x00'
Import a Model

In SQL, you can import the serialized format of an OML4Py model into an Oracle Machine Learning for SQL model with the `DBMS_DATA_MINING.IMPORT_SERMODEL` procedure. To that procedure, you pass the BLOB content from the table to which the model was exported and the name of the model to be created. The import procedure provides the ability to score the model. It does not create model views or tables that are needed for querying model details. You can use the SQL function `PREDICTION` to apply the imported model to the test data and get the prediction results.

**Example 5-3  Import a Serialized SVM Model as an OML4SQL Model in SQL**

This example retrieves the serialized content of the SVM classification model from the `svm_sermod` table. It uses the `IMPORT_SERMODEL` procedure to create a model named `my_iris_svm_classifier` with the content from the table. It also predicts test data saved in the `iris_test_data` table with the newly imported model `my_iris_svm_classifier`, and compares the prediction results with the target classes.

-- After starting SQL*Plus as the OML4Py user.
-- Import the model from the serialized content.

```sql
DECLARE
  v_blob blob;
BEGIN
  SELECT SERVAL INTO v_blob FROM "svm_sermod";
  dbms_data_mining.import_sermodel(v_blob, 'my_iris_svm_classifier');
END;
/
```

-- Set the output column format.
column TARGET_SPECIES format a15
column PREDICT_SPECIES format a15

-- Make predictions and display cases where mod(ID,3) equals 0.
SELECT ID, "Species" AS TARGET_SPECIES,
  PREDICTION(my_iris_svm_classifier USING "Sepal_Length", "Sepal_Width",
              "Petal_Length", "Petal_Width")
  AS PREDICT_SPECIES
FROM "IRIS_TEST_DATA" WHERE MOD(ID,3) = 0;

-- Drop the imported model
BEGIN
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'my_iris_svm_classifier');
END;
/

The prediction produces the following results.

<table>
<thead>
<tr>
<th>ID</th>
<th>TARGET_SPECIES</th>
<th>PREDICT_SPECIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>24</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>27</td>
<td>setosa</td>
<td>setosa</td>
</tr>
</tbody>
</table>
Model Explainability

Use the OML4Py Explainability module to identify the important features that impact a trained model’s predictions.

Machine Learning Explainability (MLX) is the process of explaining and interpreting machine learning models. The OML MLX Python module supports the ability to help better understand a model's behavior and why it makes its predictions. MLX currently provides model-agnostic explanations for classification and regression tasks where explanations treat the ML model as a black-box, instead of using properties from the model to guide the explanation.

The global feature importance explainer object is the interface to the MLX permutation importance explainer. The global feature importance explainer identifies the most important features for a given model and data set. The explainer is model-agnostic and currently supports tabular classification and regression data sets with both numerical and categorical features.

The algorithm estimates feature importance by evaluating the model's sensitivity to changes in a specific feature. Higher sensitivity suggests that the model places higher importance on that feature when making its predictions than on another feature with lower sensitivity.

For information on the `oml.GlobalFeatureImportance` class attributes and methods, invoke `help(oml.mlx.GlobalFeatureImportance)` or see Oracle Machine Learning for Python API Reference.

Example 5-4 Binary Classification

This example uses the Breast Cancer binary classification data set. Load the data set into the database and a unique case id column.

```python
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets
bc_ds = datasets.load_breast_cancer()
bcc_data = bc_ds.data.astype(float)
X = pd.DataFrame(bcc_data, columns=bc_ds.feature_names)
y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(bcc_data.shape[0]),
columns=['CASE_ID'])
```
df = oml.create(pd.concat([X, y, row_id], axis=1),
    table='BreastCancer')

Split the data set into train and test variables.

train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
    seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

Train a Random Forest model.

model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y,
    case_id='CASE_ID')
"RF accuracy score = {:.2f}".format(model.score(X_test, y_test))

Create the MLX Global Feature Importance explainer, using the binary f1 metric.

gfi = GlobalFeatureImportance(mining_function='classification',
    score_metric='f1', random_state=32,
    parallel=4)

Run the explainer to generate the global feature importance. Here we construct an explanation using the train data set and then display the explanation.

explanation = gfi.explain(model, X, y,
    case_id='CASE_ID', n_iter=10)

Drop the BreastCancer table.

oml.drop('BreastCancer')

Listing for This Example

```python
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets

>>> bc_ds = datasets.load_breast_cancer()
>>> bc_data = bc_ds.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
>>> y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]),
...                        columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1),
...                 table='BreastCancer')

>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                        seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
```
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y,
case_id='CASE_ID')
...        "RF accuracy score = {:.2f}".format(model.score(X_test,
y_test))
'RF accuracy score = 0.95'

>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                               score_metric='f1', random_state=32,
...                               parallel=4)

>>> explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)

Global Feature Importance:
[0] worst concave points: Value: 0.0263, Error: 0.0069
[1] worst perimeter: Value: 0.0077, Error: 0.0027
[2] worst radius: Value: 0.0076, Error: 0.0031
[3] worst area: Value: 0.0045, Error: 0.0037
[4] mean concave points: Value: 0.0034, Error: 0.0033
[5] worst texture: Value: 0.0017, Error: 0.0015
[6] area error: Value: 0.0012, Error: 0.0014
[7] worst concavity: Value: 0.0008, Error: 0.0008
[8] worst symmetry: Value: 0.0004, Error: 0.0007
[9] mean texture: Value: 0.0003, Error: 0.0007
[10] mean perimeter: Value: 0.0003, Error: 0.0015
[11] mean radius: Value: 0.0000, Error: 0.0000
[12] mean smoothness: Value: 0.0000, Error: 0.0000
[13] mean compactness: Value: 0.0000, Error: 0.0000
[14] mean concavity: Value: 0.0000, Error: 0.0000
[15] mean symmetry: Value: 0.0000, Error: 0.0000
[16] mean fractal dimension: Value: 0.0000, Error: 0.0000
[17] radius error: Value: 0.0000, Error: 0.0000
[18] texture error: Value: 0.0000, Error: 0.0000
[19] smoothness error: Value: 0.0000, Error: 0.0000
[20] compactness error: Value: 0.0000, Error: 0.0000
[21] concavity error: Value: 0.0000, Error: 0.0000
[22] concave points error: Value: 0.0000, Error: 0.0000
[23] symmetry error: Value: 0.0000, Error: 0.0000
[24] fractal dimension error: Value: 0.0000, Error: 0.0000
[25] worst compactness: Value: 0.0000, Error: 0.0000
[26] worst fractal dimension: Value: 0.0000, Error: 0.0000
[27] mean area: Value: -0.0001, Error: 0.0011
[28] worst smoothness: Value: -0.0003, Error: 0.0013

Example 5-5 Multi-Class Classification

This example uses the Iris multi-class classification data set. Load the data set into the
database, adding a unique case id column.

import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

iris_ds = datasets.load_iris()
iris_data = iris_ds.data.astype(float)
X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')

Split the data set into train and test variables.

train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                      seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

Train an SVM model.

model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))

Create the MLX Global Feature Importance explainer, using the f1_weighted metric.

gfi = GlobalFeatureImportance(mining_function='classification',
                            score_metric='f1_weighted',
                            random_state=32, parallel=4)

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

explanation = gfi.explain(model, X_test, y_test,
                          case_id='CASE_ID', n_iter=10)
explanation

Drop the Iris table.

oml.drop('Iris')

Listing for This Example

>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>> >>> iris_ds = datasets.load_iris()
>>> >>> iris_data = iris_ds.data.astype(float)
>>> >>> X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
>>> y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
...                        columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')

>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                        seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y,
...                                              case_id='CASE_ID')
>>> "SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
'SVM accuracy score = 0.94'

>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                                score_metric='f1_weighted',
...                                random_state=32, parallel=4)

>>> explanation = gfi.explain(model, X_test, y_test,
...                             case_id='CASE_ID', n_iter=10)

>>> explanation
Global Feature Importance:
[0] petal length (cm): Value: 0.3462, Error: 0.0824
[1] petal width (cm): Value: 0.2417, Error: 0.0687
[2] sepal width (cm): Value: 0.0926, Error: 0.0452
[3] sepal length (cm): Value: 0.0253, Error: 0.0152

>>> oml.drop('Iris')

Example 5-6  Regression

This example uses the Boston regression data set. Load the data set into the database, adding a unique case id column.

import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

boston_ds = datasets.load_boston()
boston_data = boston_ds.data
X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
y = pd.DataFrame(boston_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
                      columns=['CASE_ID'])

df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')

Split the data set into train and test variables.

train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
Train a Neural Network regression model.

```python
model = oml.algo.nn(mining_function='regression',
                      ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the $r^2$ metric.

```python
gfi = GlobalFeatureImportance(mining_function='regression',
                               score_metric='r2', random_state=32,
                               parallel=4)
```

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

```python
explanation = gfi.explain(model, df, 'TARGET',
                          case_id='CASE_ID', n_iter=10)
explanation
```

Drop the Boston table.

```python
oml.drop('Boston')
```

**Listing for This Example**

```python
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets

>>> boston_ds = datasets.load_boston()
>>> boston_data = boston_ds.data
>>> X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
>>> y = pd.DataFrame(boston_ds.target, columns=['TARGET'],
                      columns=['CASE_ID'])
>>> row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
                         columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')

>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                           seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> model = oml.algo.nn(mining_function='regression',
                      ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
'NN R^2 score = 0.85'

>>> gfi = GlobalFeatureImportance(mining_function='regression',
                               score_metric='r2', random_state=32,
                               parallel=4)
```
>>> explanation = gfi.explain(model, df, 'TARGET',
...                            case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] LSTAT: Value: 0.7686, Error: 0.0513
[1] RM: Value: 0.5734, Error: 0.0475
[2] CRIM: Value: 0.5131, Error: 0.0345
[3] DIS: Value: 0.4170, Error: 0.0632
[4] NOX: Value: 0.2592, Error: 0.0206
[5] AGE: Value: 0.2083, Error: 0.0212
[6] RAD: Value: 0.1956, Error: 0.0188
[7] INDUS: Value: 0.1792, Error: 0.0199
[8] B: Value: 0.0982, Error: 0.0146
[9] PTRATIO: Value: 0.0822, Error: 0.0069
[10] TAX: Value: 0.0566, Error: 0.0139
[11] ZN: Value: 0.0397, Error: 0.0081
[12] CHAS: Value: 0.0125, Error: 0.0045

>>> oml.drop('Boston')

Attribute Importance

The oml.ai class computes the relative attribute importance, which ranks attributes according to their significance in predicting a classification or regression target.

The oml.ai class uses the Minimum Description Length (MDL) algorithm to calculate attribute importance. MDL assumes that the simplest, most compact representation of the data is the best and most probable explanation of the data.

You can use methods of the oml.ai class to compute the relative importance of predictor variables when predicting a response variable.

Note:
Oracle Machine Learning does not support the scoring operation for oml.ai.

The results of oml.ai are the attributes of the build data ranked according to their predictive influence on a specified target attribute. You can use the ranking and the measure of importance for selecting attributes.

For information on the oml.ai class attributes and methods, invoke help(oml.ai) or see Oracle Machine Learning for Python API Reference.

See Also:
- About Model Settings
- Shared Settings
Example 5-7   Ranking Attribute Significance with oml.ai

This example creates the \(x\) and \(y\) variables using the iris data set. It then creates the persistent database table IRIS and the oml.DataFrame object oml_iris as a proxy for the table.

This example demonstrates the use of various methods of the oml.ai class.

import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
    {0: 'setosa', 1: 'versicolor',
    2:'virginica'}[x], iris.target)),
    columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Specify settings.
setting = {'ODMS_SAMPLING':'ODMS_SAMPLING_DISABLE'}

# Create an AI model object.
ai_mod = oml.ai(**setting)

# Fit the AI model according to the training data and parameter
# settings.
ai_mod = ai_mod.fit(train_x, train_y)

# Show the model details.
ai_mod

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
... x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

>>> # Specify settings.
... setting = {'ODMS_SAMPLING':ODMS_SAMPLING_DISABLE}

>>> # Create an AI model object.
... ai_mod = oml.ai(**setting)

>>> # Fit the AI model according to the training data and parameter
... # settings.
... ai_mod = ai_mod.fit(train_x, train_y)

>>> # Show the model details.
... ai_mod

Algorithm Name: Attribute Importance

Mining Function: ATTRIBUTE_IMPORTANCE

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME</td>
</tr>
<tr>
<td>1</td>
<td>ALGO_AI_MDL</td>
</tr>
<tr>
<td>2</td>
<td>ODMS_DETAILS</td>
</tr>
<tr>
<td>3</td>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
</tr>
<tr>
<td>4</td>
<td>ODMS_SAMPLING</td>
</tr>
<tr>
<td>5</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>6</td>
<td>PREP_AUTO</td>
</tr>
<tr>
<td>7</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUM_ROWS</td>
</tr>
<tr>
<td>1</td>
<td>104</td>
</tr>
</tbody>
</table>

Attributes:

Petal_Length
Association Rules

The `oml.ar` class implements the Apriori algorithm to find frequent itemsets and association rules, all as part of an association model object.

The Apriori algorithm is efficient and scales well with respect to the number of transactions, number of items, and number of itemsets and rules produced.

Use the `oml.ar` class to identify frequent itemsets within large volumes of transactional data, such as in market basket analysis. The results of an association model are the rules that identify patterns of association within the data.

An association rule identifies a pattern in the data in which the appearance of a set of items in a transactional record implies another set of items. The groups of items used to form rules must pass a minimum threshold according to how often they occur (the `support` of the rule) and how often the consequent follows the antecedent (the `confidence` of the rule). Association models generate all rules that have support and confidence greater than user-specified thresholds.

Oracle Machine Learning does not support the scoring operation for association modeling.

For information on the `oml.ar` class attributes and methods, invoke `help(oml.ar)` or see `Oracle Machine Learning for Python API Reference`.

Settings for an Association Rules Model

The following table lists the settings applicable to association rules models.

<table>
<thead>
<tr>
<th>variable</th>
<th>importance</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Width</td>
<td>0.615851</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Length</td>
<td>0.362519</td>
</tr>
<tr>
<td>2</td>
<td>Sepal_Length</td>
<td>0.042751</td>
</tr>
<tr>
<td>3</td>
<td>Sepal_Width</td>
<td>-0.155867</td>
</tr>
</tbody>
</table>
### Table 5-2  Association Rules Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSO_ABS_ERROR</td>
<td>0 &lt; ASSO_ABS_ERROR ≤ MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)</td>
<td>Specifies the absolute error for the association rules sampling. A smaller value of ASSO_ABS_ERROR obtains a larger sample size that gives accurate results but takes longer to compute. Set a reasonable value for ASSO_ABS_ERROR, such as the default value, to avoid too large a sample size. The default value is ( 0.5 \times \max(\text{ASSO_MIN}_\text{SUPPORT}, \text{ASSO_MIN}_\text{CONFIDENCE}) ).</td>
</tr>
<tr>
<td>ASSO_AGGREGATES</td>
<td>NULL</td>
<td>Specifies the columns to aggregate. It is a comma separated list of strings containing the names of the columns for aggregation. The number of columns in the list must be ( \leq 10 ). You can set ASSO_AGGREGATES if you have specified a column name with ODMS_ITEM_ID_COLUMN_NAME. The data table must have valid column names such as ITEM_ID and CASE_ID which are derived from ODMS_ITEM_ID_COLUMN_NAME. An item value is not mandatory. The default value is NULL. For each item, you may supply several columns to aggregate. However, doing so requires more memory to buffer the extra data and also affects performance because of the larger input data set and increased operations.</td>
</tr>
<tr>
<td>ASSO_ANT_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules for the antecedent: it is a comma separated list of strings, at least one of which must appear in the antecedent part of each reported association rule. The default value is NULL.</td>
</tr>
<tr>
<td>ASSO_ANT_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules for the antecedent: it is a comma separated list of strings, none of which can appear in the antecedent part of each reported association rule. The default value is NULL.</td>
</tr>
<tr>
<td>ASSO_CONF_LEVEL</td>
<td>0 ≤ ASSO_CONF_LEVEL ≤ 1</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 5-2  (Cont.) Association Rules Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSO_CONS_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules for the consequent: it is a comma separated list of strings, at least one of which must appear in the consequent part of each reported association rule. The default value is NULL.</td>
</tr>
<tr>
<td>ASSO_CONS_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules for the consequent: it is a comma separated list of strings, none of which can appear in the consequent part of a reported association rule. You can use the excluding rule to reduce the data that must be stored, but you may be required to build extra models for executing different Including or Excluding Rules. The default value is NULL.</td>
</tr>
<tr>
<td>ASSO_EX_RULES</td>
<td>NULL</td>
<td>Sets Excluding Rules applied for each association rule: it is a comma separated list of strings that cannot appear in an association rule. No rule can contain any item in the list. The default value is NULL.</td>
</tr>
<tr>
<td>ASSO_IN_RULES</td>
<td>NULL</td>
<td>Sets Including Rules applied for each association rule: it is a comma separated list of strings, at least one of which must appear in each reported association rule, either as antecedent or as consequent. The default value NULL, which specifies that filtering is not applied.</td>
</tr>
<tr>
<td>ASSO_MAX_RULE_LENGTH</td>
<td>TO_CHAR( 2&lt;= numeric_expr &lt;=20)</td>
<td>Maximum rule length for association rules. The default value is 4.</td>
</tr>
<tr>
<td>ASSO_MIN_CONFIDENCE</td>
<td>TO_CHAR( 0&lt;= numeric_expr &lt;=1)</td>
<td>Minimum confidence for association rules. The default value is 0.1.</td>
</tr>
<tr>
<td>ASSO_MIN_REV_CONFIDENCE</td>
<td>TO_CHAR( 0&lt;= numeric_expr &lt;=1)</td>
<td>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 value is 0.</td>
</tr>
<tr>
<td>Setting Name</td>
<td>Setting Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ASSO_MIN_SUPPORT</td>
<td>TO_CHAR( 0&lt;= numeric_expr &lt;=1)</td>
<td>Minimum support for association rules. The default value is 0.1.</td>
</tr>
<tr>
<td>ASSO_MIN_SUPPORT_INT</td>
<td>TO_CHAR( 0&lt;= numeric_expr &lt;=1)</td>
<td>Minimum absolute support that each rule must satisfy. The value must be an integer. The default value is 1.</td>
</tr>
<tr>
<td>ASSO_CONS_EX_RULES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODMS_ITEM_ID_COLUMN_NAME</td>
<td>column_name</td>
<td>The name of a column that contains the items in a transaction. When you specify this setting, the algorithm expects the data to be presented in native transactional format, consisting of two columns: • Case ID, either categorical or numeric • Item ID, either categorical or numeric</td>
</tr>
<tr>
<td>ODMS_ITEM_VALUE_COLUMN_NAME</td>
<td>column_name</td>
<td>The name of a column that contains a value associated with each item in a transaction. Use this setting only when you have specified a value for ODMS_ITEM_ID_COLUMN_NAME, indicating that the data is presented in native transactional format. If you also use ASSO_AGGREGATES, 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 Name</td>
</tr>
</tbody>
</table>
Example 5-8 Using the oml.ar Class

This example uses methods of the \texttt{oml.ar} class.

```python
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length','Sepal_Width',
    'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
    {0: 'setosa', 1: 'versicolor',
    2:'virginica'}[x], iris.target)),
    columns = ['Species']))

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training data.
train_dat = oml.sync(table = 'IRIS')

# Specify settings.
setting = {'asso_min_support':'0.1', 'asso_min_confidence':'0.1'}

# Create an AR model object.
ar_mod = oml.ar(**setting)

# Fit the model according to the training data and parameter settings.
ar_mod = ar_mod.fit(train_dat)

# Show details of the model.
ar_mod
```

Listing for This Example

```python
>>> import pandas as pd
>>> from sklearn import datasets
```
>>> import oml

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
... x = pd.DataFrame(iris.data,
...                   columns = ['Sepal_Length', 'Sepal_Width',
...                               'Petal_Length', 'Petal_Width'])
... y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                   columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training data.
... train_dat = oml.sync(table = 'IRIS')

>>> # Specify settings.
... setting = {'asso_min_support': 0.1, 'asso_min_confidence': 0.1}

>>> # Create an AR model object.
... ar_mod = oml.ar(**setting)

>>> # Fit the model according to the training data and parameter
... # settings.
... ar_mod = ar_mod.fit(train_dat)

>>> # Show details of the model.
... ar_mod

Algorithm Name: Association Rules

Mining Function: ASSOCIATION

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_APRIORI_ASSOCIATION_RULES</td>
</tr>
<tr>
<td>ASSO_MAX_RULE_LENGTH</td>
<td>4</td>
</tr>
<tr>
<td>ASSO_MIN_CONFIDENCE</td>
<td>0.1</td>
</tr>
<tr>
<td>ASSO_MIN_REV_CONFIDENCE</td>
<td>0</td>
</tr>
<tr>
<td>ASSO_MIN_SUPPORT</td>
<td>0.1</td>
</tr>
<tr>
<td>ASSO_MIN_SUPPORT_INT</td>
<td>1</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
</table>
Decision Tree

The `oml.dt` class uses the Decision Tree algorithm for classification.

Decision Tree models are classification models that contain axis-parallel rules. A rule is a conditional statement that can be understood by humans and may be used within a database to identify a set of records.

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.

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. The `oml.dt` class offers two homogeneity metrics, gini and entropy, for calculating the splits. The default metric is gini.

For information on the `oml.dt` class attributes and methods, invoke `help(oml.dt)` or see Oracle Machine Learning for Python API Reference.
Settings for a Decision Tree Model

The following table lists settings that apply to Decision Tree models.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLAS_COST_TABLE_NAME         | table_name         | The name of a table that stores a cost matrix for the algorithm to use in building and applying the model. The cost matrix specifies the costs associated with misclassifications.  
   The cost matrix table is user-created.  
   The following are the column requirements for the table.  
   - Column Name: ACTUAL_TARGET_VALUE  
     Data Type: Valid target data type  
   - Column Name: PREDICTED_TARGET_VALUE  
     Data Type: Valid target data type  
   - Column Name: COST  
     Data Type: NUMBER |
| CLAS_MAX_SUP_BINS            | 2 <= a number <= | Specifies the maximum number of bins for each attribute.  
   2147483647                                                            |
| CLAS_WEIGHTS_BALANCED        | ON                 | Indicates whether 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.                  |
|                             | OFF                |                                                                                                                                           |
| TREE_IMPURITY_METRIC        | TREE_IMPURITY_ENTROPY | Tree impurity metric for a Decision Tree model.  
   Tree algorithms seek the best test question for splitting data at each node.  
   The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with 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. |
Table 5-3  (Cont.) Decision Tree Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE_TERM_MAX_DEPTH</td>
<td>$2 \leq a \text{ number} \leq 100$</td>
<td>Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). The default is 7.</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>$0 &lt; a \text{ number} \leq 10$</td>
<td>The minimum number of training rows in a node expressed as a percentage of the rows in the training data. The default value is 0.05, indicating 0.05%.</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>$0 &lt; a \text{ number} \leq 20$</td>
<td>Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>A \text{ number} $\geq 0$</td>
<td>Minimum number of rows in a node. The default value is 10.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>A \text{ number} $&gt; 1$</td>
<td>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. The default value is 20.</td>
</tr>
</tbody>
</table>

See Also:
- About Model Settings
- Shared Settings

Example 5-9  Using the oml.dt Class

This example demonstrates the use of various methods of the oml.dt class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length','Sepal_Width',
                'Petal_Length','Petal_Width'])

y = pd.DataFrame(list(map(lambda x:
                        {0: 'setosa', 1: 'versicolor',
                         2:'virginica'}[x], iris.target)),
```
try:
    oml.drop('COST_MATRIX')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0],
               ['setosa', 'virginica', 0.2],
               ['setosa', 'versicolor', 0.8],
               ['virginica', 'virginica', 0],
               ['virginica', 'setosa', 0.5],
               ['virginica', 'versicolor', 0.5],
               ['versicolor', 'versicolor', 0],
               ['versicolor', 'setosa', 0.4],
               ['versicolor', 'virginica', 0.6]]

# Specify settings.
setting = {'TREE_TERM_MAX_DEPTH':2'

# Create a DT model object.
dt_mod = oml.dt(**setting)

# Fit the DT model according to the training data and parameter
# settings.
dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Use the model to make predictions on the test data.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                             'Sepal_Width',
                                             'Petal_Length',
                                             'Species']])

# Return the prediction probability.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                             'Sepal_Width',
                                             'Species']])
proba = True

# Make predictions and return the probability for each class
# on new data.
dt_mod.predict_proba(test_dat.drop('Species'),
                     supplemental_cols = test_dat[:,
                     ['Sepal_Length',
                      'Species']].sort_values(by = ['Sepal_Length',
                      'Species']))

dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> try:
...    oml.drop('COST_MATRIX')
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

>>> # Create a cost matrix table in the database.
... cost_matrix = [[setosa, setosa, 0],
...                [setosa, virginica, 0.2],
...                [setosa, versicolor, 0.8],
...                [virginica, virginica, 0],
...                [virginica, setosa, 0.5],
...                [virginica, versicolor, 0.5],
...                [versicolor, versicolor, 0],
...                [versicolor, setosa, 0.4],
...                [versicolor, virginica, 0.6]]

>>> cost_matrix = oml.create(}
... pd.DataFrame(cost_matrix, columns = ['ACTUAL_TARGET_VALUE', 'PREDICTED_TARGET_VALUE', 'COST']),
... table = 'COST_MATRIX')

>>> # Specify settings.
... setting = {'TREE_TERM_MAX_DEPTH': '2'}

>>> # Create a DT model object.
... dt_mod = oml.dt(**setting)

>>> # Fit the DT model according to the training data and parameter settings.
... dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_DECISION_TREE</td>
</tr>
<tr>
<td>CLAS_COST_TABLE_NAME</td>
<td>&quot;OML_USER&quot;.&quot;COST_MATRIX&quot;</td>
</tr>
<tr>
<td>CLAS_MAX_SUP_BINS</td>
<td>32</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
<tr>
<td>TREE_IMPURITY_METRIC</td>
<td>TREE_IMPURITY_GINI</td>
</tr>
<tr>
<td>TREE_TERM_MAX_DEPTH</td>
<td>2</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>.05</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>.1</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>10</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>20</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_ROWS</td>
<td>104</td>
</tr>
</tbody>
</table>

Attributes:

Petal_Length
Petal_Width

Partition: NO

Distributions:

<table>
<thead>
<tr>
<th>NODE_ID</th>
<th>TARGET_VALUE</th>
<th>TARGET_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>setosa</td>
<td>36</td>
</tr>
<tr>
<td>1</td>
<td>versicolor</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>virginica</td>
<td>33</td>
</tr>
</tbody>
</table>
3 1 setosa 36
4 2 versicolor 35
5 2 virginica 33

Nodes:

<table>
<thead>
<tr>
<th>parent</th>
<th>node.id</th>
<th>row.count</th>
<th>prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>2</td>
<td>68</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
<td>0</td>
<td>104</td>
</tr>
</tbody>
</table>

split

0 (Petal_Length <=(2.4500000000000002E+000))
1 (Petal_Length > (2.4500000000000002E+000))
2 None

surrogate

0 Petal_Width <=(8.0000000000000004E-001))
1 Petal_Width > (8.0000000000000004E-001))
2 None

full.splits

0 (Petal_Length <=(2.4500000000000002E+000))
1 (Petal_Length > (2.4500000000000002E+000))
2 (None)

# Use the model to make predictions on the test data.
... dt_mod.predict(test_dat.drop('Species'),
... supplemental_cols = test_dat[:, ['Sepal_Length',
... 'Sepal_Width',
... 'Species']])

Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0 4.9          3.0           1.4      setosa      setosa
1 4.9          3.1           1.5      setosa      setosa
2 4.8          3.4           1.6      setosa      setosa
3 5.8          4.0           1.2      setosa      setosa
...          ...           ...      ...        ...
44 6.7          3.3           5.7      virginica  versicolor
45 6.7          3.0           5.2      virginica  versicolor
46 6.5          3.0           5.2      virginica  versicolor
47 5.9          3.0           5.1      virginica  versicolor

# Return the prediction probability.
... dt_mod.predict(test_dat.drop('Species'),
... supplemental_cols = test_dat[:, ['Sepal_Length',
... 'Sepal_Width',
... 'Species']],
... proba = True)

Sepal_Length  Sepal_Width  Species  PREDICTION  PROBABILITY
0 4.9          3.0      setosa      setosa     1.000000
1 4.9          3.1      setosa      setosa     1.000000
2 4.8          3.4      setosa      setosa     1.000000
3 5.8          4.0      setosa      setosa     1.000000
...          ...        ...      ...        ...
```python
>>> # Make predictions and return the probability for each class
>>> # on new data.
>>> dt_mod.predict_proba(test_dat.drop('Species'),
... supplemental_cols = test_dat[:,
... ['Sepal_Length',
... 'Species']].sort_values(by =
[ 'Sepal_Length',
... 'Species'])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Species</th>
<th>PROBABILITY_OF_SETOSA</th>
<th>PROBABILITY_OF_VERSICOLOR</th>
<th>PROBABILITY_OF_VIRGINICA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.4</td>
<td>setosa</td>
<td>1.0</td>
<td>0.000000</td>
</tr>
<tr>
<td>1</td>
<td>4.4</td>
<td>setosa</td>
<td>1.0</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>setosa</td>
<td>1.0</td>
<td>0.000000</td>
</tr>
<tr>
<td>3</td>
<td>4.8</td>
<td>setosa</td>
<td>1.0</td>
<td>0.000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>6.7</td>
<td>virginica</td>
<td>0.0</td>
<td>0.514706</td>
</tr>
<tr>
<td>43</td>
<td>6.9</td>
<td>versicolor</td>
<td>0.0</td>
<td>0.514706</td>
</tr>
<tr>
<td>44</td>
<td>6.9</td>
<td>virginica</td>
<td>0.0</td>
<td>0.514706</td>
</tr>
<tr>
<td>45</td>
<td>7.0</td>
<td>versicolor</td>
<td>0.0</td>
<td>0.514706</td>
</tr>
</tbody>
</table>

>>> dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.645833
```

Expectation Maximization

The `oml.em` class uses the Expectation Maximization (EM) algorithm to create a clustering model.

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.

For information on the `oml.em` class methods, invoke `help(oml.em)` or see Oracle Machine Learning for Python API Reference.

Settings for an Expectation Maximization Model

The following table lists settings for data preparation and analysis for EM models.
<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_ATTRIBUTE_FILTER</td>
<td>EMCS_ATTR_FILTER_ENABLE EMCS_ATTR_FILTER_DISABLE</td>
<td>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. The default value is system-determined.</td>
</tr>
<tr>
<td>EMCS_MAX_NUM_ATTR_2D</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Maximum number of correlated attributes to include in the model. Note: This setting applies only to attributes that are not nested (2D). The default value is 50.</td>
</tr>
</tbody>
</table>
# Expectation Maximization Settings for Data Preparation and Analysis

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_NUM_DISTRIBUTION</td>
<td>EMCS_NUM_DISTR_BERNOULLI EMCS_NUM_DISTR_GAUSSIAN EMCS_NUM_DISTR_SYSTEM</td>
<td>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. The default value is EMCS_NUM_DISTR_SYSTEM.</td>
</tr>
<tr>
<td>EMCS_NUM_EQUIWIDTH_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. The default value is 11.</td>
</tr>
<tr>
<td>EMCS_NUM_PROJECTIONS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Specifies the number of projections to use for each nested column. If a column has fewer distinct attributes than the specified number of projections, then the data is not projected. The setting applies to all nested columns. The default value is 50.</td>
</tr>
<tr>
<td>EMCS_NUM_QUANTILE_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Specifies the number of quantile bins to use for modeling numeric columns with multivalued Bernoulli distributions. The default value is system-determined.</td>
</tr>
<tr>
<td>EMCS_NUM_TOPN_BINS</td>
<td>TO_CHAR(1 &lt; numeric_expr &lt;= 255)</td>
<td>Specifies the number of top-N bins to use for modeling categorical columns with multivalued Bernoulli distributions. The default value is system-determined.</td>
</tr>
</tbody>
</table>

The following table lists settings for learning for EM models.

---

Oracle
Table 5-5  Expectation Maximization Settings for Learning

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_CONVERGENCE_CRITERION</td>
<td>EMCS_CONV_CRIT_HELDASIDE, EMCS_CONV_CRIT_BIC</td>
<td>The convergence criterion for EM. The convergence criterion may be based on a held-aside data set or it may be Bayesian Information Criterion. The default value is system determined.</td>
</tr>
<tr>
<td>EMCS_LOGLIKE_IMPROVEMENT</td>
<td>TO_CHAR( 0 &lt; numeric_expr &lt; 1)</td>
<td>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.</td>
</tr>
<tr>
<td>EMCS_MODEL_SEARCH</td>
<td>EMCS_MODEL_SEARCH_ENABLE, EMCS_MODEL_SEARCH_DISABLE</td>
<td>Enables model search in EM where different model sizes are explored and the best size is selected. The default value is EMCS_MODEL_SEARCH_DISABLE.</td>
</tr>
<tr>
<td>EMCS_NUM_COMPONENTS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>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. The number of components must be greater than or equal to the number of clusters. The default value is 20.</td>
</tr>
<tr>
<td>EMCS_NUM_ITERATIONS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>Specifies the maximum number of iterations in the EM algorithm. The default value is 100.</td>
</tr>
<tr>
<td>EMCS_RANDOM_SEED</td>
<td>Non-negative integer</td>
<td>Controls the seed of the random generator used in EM. The default value is 0.</td>
</tr>
<tr>
<td>EMCS_REMOVE_COMPONENTS</td>
<td>EMCS_REMOVE_COMPS_ENABLE, EMCS_REMOVE_COMPS_DISABLE</td>
<td>Allows the EM algorithm to remove a small component from the solution. The default value is EMCS_REMOVE_COMPS_ENABLE.</td>
</tr>
</tbody>
</table>

The following table lists the settings for component clustering for EM models.
### Table 5-6  Expectation Maximization Settings for Component Clustering

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUS_NUM_CLUSTERS</td>
<td>TO_CHAR(numeric_expr &gt;= 1)</td>
<td>The maximum number of leaf clusters generated by the algorithm. The algorithm may return fewer clusters than the specified number, depending on the data, but it cannot return more clusters than the number of components, which is governed by algorithm-specific settings. (See Table 5-5.) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, then the number of clusters equals the number of components. The default value is system-determined.</td>
</tr>
</tbody>
</table>
| 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.  
When component clustering is enabled, model scoring through the SQL CLUSTER function produces assignments to the higher level clusters. When clustering is disabled, the CLUSTER function produces assignments to the original components.  
The default value 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.  
A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out.  
The default value is 2. |
### Table 5-6  (Cont.) Expectation Maximization Settings for Component Clustering

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_LINKAGE_FUNCTION</td>
<td>EMCS_LINKAGE_SINGLE</td>
<td>Allows the specification of a linkage function for the agglomerative clustering step.</td>
</tr>
<tr>
<td></td>
<td>EMCS_LINKAGE_AVERAGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EMCS_LINKAGE_COMPLETE</td>
<td></td>
</tr>
<tr>
<td>EMCS_LINKAGE_FUNCTION</td>
<td></td>
<td><strong>EMCS_LINKAGE_SINGLE</strong> uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>EMCS_LINKAGE_AVERAGE</strong> uses the average distance within the branch. There is less chaining effect and the clusters are more compact.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>EMCS_LINKAGE_COMPLETE</strong> uses the maximum distance within the branch. The clusters are smaller and require strong component overlap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is <strong>EMCS_LINKAGE_SINGLE</strong>.</td>
</tr>
</tbody>
</table>

The following table lists the settings for cluster statistics for EM models.

### Table 5-7  Expectation Maximization Settings for Cluster Statistics

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_CLUSTER_STATISTICS</td>
<td>EMCS_CLUS_STATS_ENABLE</td>
<td>Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced. The default value is <strong>EMCS_CLUS_STATS_ENABLE</strong>.</td>
</tr>
<tr>
<td>EMCS_CLUSTER_STATISTICS</td>
<td>EMCS_CLUS_STATS_DISABLE</td>
<td></td>
</tr>
<tr>
<td>EMCS_MIN_PCT_ATTRIB_SUPPORT</td>
<td>TO_CHAR( 0 &lt; numeric_expr &lt; 1)</td>
<td>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 default value is 0.1.</td>
</tr>
</tbody>
</table>
Example 5-10  Using the oml.em Class

This example creates an EM model and uses some of the methods of the `oml.em` class.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                            'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
                                      2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
setting = {'emcs_num_iterations': 100}

# Create an EM model object
em_mod = oml.em(n_clusters = 2, **setting)

# Fit the EM model according to the training data and parameter
# settings.
em_mod = em_mod.fit(train_dat)

# Show details of the model.
em_mod

# Use the model to make predictions on the test data.
em_mod.predict(test_dat)

# Make predictions and return the probability for each class
# on new data.
em_mod.predict_proba(test_dat,
                      supplemental_cols = test_dat[:, [
                                                  'Sepal_Length', 'Sepal_Width',
                                                  'Petal_Length']].sort_values(by = ['Sepal_Length',
                                                                                   'Sepal_Width', 'Petal_Length'],
                                                                                  ascending = False)
                      )
```

Chapter 5
Expectation Maximization

5-38
# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal.Length', 'Sepal.Width',
...                             'Petal.Length', 'Petal.Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]

>>> # Specify settings.
... setting = {'emcs_num_iterations': 100}

>>> # Create an EM model object.
... em_mod = oml.em(n_clusters = 2, **setting)

>>> # Fit the EM model according to the training data and parameter
... # settings.
... em_mod = em_mod.fit(train_dat)

>>> # Show details of the model.
... em_mod
```

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME ALGOEXPECTATION_MAXIMIZATION</td>
</tr>
</tbody>
</table>
Chapter 5
Expectation Maximization

## Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMCS_ATTRIBUTE_FILTER</td>
<td>EMCS_ATTR_FILTER_DISABLE</td>
</tr>
<tr>
<td>EMCS_CONVERGENCE_CRITERION</td>
<td>EMCS_CONV_CRIT_BIC</td>
</tr>
<tr>
<td>EMCS_NUM_QUANTILE_BINS</td>
<td>3</td>
</tr>
<tr>
<td>EMCS_NUM_TOPN_BINS</td>
<td>3</td>
</tr>
</tbody>
</table>

## Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERGED</td>
<td>YES</td>
</tr>
<tr>
<td>LOGLIKELIHOOD</td>
<td>-2.10044</td>
</tr>
<tr>
<td>NUM_CLUSTERS</td>
<td>2</td>
</tr>
<tr>
<td>NUM_COMPONENTS</td>
<td>8</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>104</td>
</tr>
<tr>
<td>RANDOM_SEED</td>
<td>0</td>
</tr>
<tr>
<td>REMOVED_COMPONENTS</td>
<td>12</td>
</tr>
</tbody>
</table>

## Attributes:
- Petal_Length
- Petal_Width
- Sepal_Length
- Sepal_Width
- Species

## Partition: NO

## Clusters:

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>CLUSTER_NAME</th>
<th>RECORD_COUNT</th>
<th>PARENT</th>
<th>TREE_LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>NaN</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1.0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEFT_CHILD_ID</th>
<th>RIGHT_CHILD_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Taxonomy:

<table>
<thead>
<tr>
<th>PARENT_CLUSTER_ID</th>
<th>CHILD_CLUSTER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Centroids:

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>MEAN</th>
<th>MODE_VALUE</th>
<th>VARIANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Length</td>
<td>3.721154</td>
<td>None</td>
<td>3.234694</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Width</td>
<td>1.155769</td>
<td>None</td>
<td>0.567539</td>
</tr>
<tr>
<td>2</td>
<td>Sepal_Length</td>
<td>5.831731</td>
<td>None</td>
<td>0.753255</td>
</tr>
<tr>
<td>3</td>
<td>Sepal_Width</td>
<td>3.074038</td>
<td>None</td>
<td>0.221358</td>
</tr>
<tr>
<td>4</td>
<td>Species</td>
<td>NaN</td>
<td>setosa</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>Petal_Length</td>
<td>4.902941</td>
<td>None</td>
<td>0.860588</td>
</tr>
<tr>
<td>6</td>
<td>Petal_Width</td>
<td>1.635294</td>
<td>None</td>
<td>0.191572</td>
</tr>
<tr>
<td>7</td>
<td>Sepal_Length</td>
<td>6.266176</td>
<td>None</td>
<td>0.545555</td>
</tr>
<tr>
<td>8</td>
<td>Sepal_Width</td>
<td>2.854412</td>
<td>None</td>
<td>0.128786</td>
</tr>
<tr>
<td>9</td>
<td>Species</td>
<td>NaN</td>
<td>versicolor</td>
<td>NaN</td>
</tr>
<tr>
<td>10</td>
<td>Petal_Length</td>
<td>1.488889</td>
<td>None</td>
<td>0.033016</td>
</tr>
<tr>
<td>11</td>
<td>Petal_Width</td>
<td>0.250000</td>
<td>None</td>
<td>0.012857</td>
</tr>
<tr>
<td>12</td>
<td>Sepal_Length</td>
<td>5.011111</td>
<td>None</td>
<td>0.113016</td>
</tr>
<tr>
<td>13</td>
<td>Sepal_Width</td>
<td>3.488889</td>
<td>None</td>
<td>0.134159</td>
</tr>
<tr>
<td>14</td>
<td>Species</td>
<td>NaN</td>
<td>setosa</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Leaf Cluster Counts:

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
</tr>
<tr>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>

Attribute Importance:

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_IMPORTANCE_VALUE</th>
<th>ATTRIBUTE_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petal_Length</td>
<td>0.558311</td>
<td>2</td>
</tr>
<tr>
<td>Petal_Width</td>
<td>0.556300</td>
<td>3</td>
</tr>
<tr>
<td>Sepal_Length</td>
<td>0.469978</td>
<td>4</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td>0.196211</td>
<td>5</td>
</tr>
<tr>
<td>Species</td>
<td>0.612463</td>
<td>1</td>
</tr>
</tbody>
</table>

Components:

<table>
<thead>
<tr>
<th>COMPONENT_ID</th>
<th>CLUSTER_ID</th>
<th>PRIOR_PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
Cluster Hists:

<table>
<thead>
<tr>
<th>cluster.id</th>
<th>variable</th>
<th>bin.id</th>
<th>lower.bound</th>
<th>upper.bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Length</td>
<td>1</td>
<td>1.00</td>
<td>1.59</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Length</td>
<td>2</td>
<td>1.59</td>
<td>2.18</td>
</tr>
<tr>
<td>2</td>
<td>Petal_Length</td>
<td>3</td>
<td>2.18</td>
<td>2.77</td>
</tr>
<tr>
<td>3</td>
<td>Petal_Length</td>
<td>4</td>
<td>2.77</td>
<td>3.36</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>137</td>
<td>Sepal_Width</td>
<td>11</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>138</td>
<td>Species;'Other'</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>139</td>
<td>Species;setosa</td>
<td>2</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>140</td>
<td>Species;versicolor</td>
<td>3</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

label count

<table>
<thead>
<tr>
<th>label</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>137</td>
<td>0</td>
</tr>
<tr>
<td>138</td>
<td>0</td>
</tr>
<tr>
<td>139</td>
<td>36</td>
</tr>
<tr>
<td>140</td>
<td>0</td>
</tr>
</tbody>
</table>

[141 rows x 7 columns]

Rules:

<table>
<thead>
<tr>
<th>cluster.id</th>
<th>rhs.support</th>
<th>rhs.conf</th>
<th>lhr.support</th>
<th>lhs.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>104</td>
<td>1.000000</td>
<td>93</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>104</td>
<td>1.000000</td>
<td>93</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>104</td>
<td>1.000000</td>
<td>99</td>
</tr>
<tr>
<td>Petal_Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>104</td>
<td>1.000000</td>
<td>99</td>
</tr>
<tr>
<td>Petal_Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>26</td>
<td>3</td>
<td>36</td>
<td>0.346154</td>
<td>36</td>
</tr>
<tr>
<td>Petal_Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>36</td>
<td>0.346154</td>
<td>36</td>
</tr>
<tr>
<td>Sepal_Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>36</td>
<td>0.346154</td>
<td>36</td>
</tr>
<tr>
<td>Sepal_Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>3</td>
<td>36</td>
<td>0.346154</td>
<td>36</td>
</tr>
<tr>
<td>Species</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

lhs.var.support  lhs.var.conf              predicate
|            |            |              |              |
| 0          | 93         | 0.400000     | Sepal_Width  <= 3.92 |
| 1          | 93         | 0.400000     | Sepal_Width  > 2.48  |
[30 rows x 9 columns]

```python
>>> # Use the model to make predictions on the test data.
>>> em_mod.predict(test_dat)
   CLUSTER_ID
0  3
1  3
2  3
3  3
... ...
42  2
43  2
44  2
45  2

>>> # Make predictions and return the probability for each class.
>>> # on new data.
>>> em_mod.predict_proba(test_dat,
...                     supplemental_cols = test_dat[:,
...                      ['Sepal_Length', 'Sepal_Width',
...                      'Petal_Length']].sort_values(by = ['Sepal_Length',
...                      'Sepal_Width', 'Petal_Length',
...                      'PROBABILITY_OF_2', 'PROBABILITY_OF_3']))
    Sepal_Length  Sepal_Width  Petal_Length  PROBABILITY_OF_2  \
   0  4.4  3.0  1.3  4.680788e-20
   1  4.4  3.2  1.3  1.052071e-20
   2  4.5  2.3  1.3  7.751240e-06
   3  4.8  3.4  1.6  5.363418e-19
   4  ...
43  6.9  3.1  4.9  1.000000e+00
44  6.9  3.1  5.4  1.000000e+00
45  7.0  3.2  4.7  1.000000e+00

   PROBABILITY_OF_3
   0  1.000000e+00
   1  1.000000e+00
   2  9.999922e-01
   3  1.000000e+00
   4  ...
43  3.295578e-97
44  6.438740e-137
45  3.853925e-89

>>> # Change the random seed and refit the model.
>>> em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ALGO_NAME</td>
<td>ALGO_EXPECTATION_MAXIMIZATION</td>
</tr>
<tr>
<td>1 CLUS_NUM_CLUSTERS</td>
<td>2</td>
</tr>
<tr>
<td>2 EMCS_CLUSTER_COMPONENTS</td>
<td>EMCS_CLUSTER_COMP_ENABLE</td>
</tr>
<tr>
<td>3 EMCS_CLUSTER_STATISTICS</td>
<td>EMCS_CLUS_STATS_ENABLE</td>
</tr>
<tr>
<td>4 EMCS_CLUSTER_THRESH</td>
<td>2</td>
</tr>
<tr>
<td>5 EMCS_LINKAGE_FUNCTION</td>
<td>EMCS_LINKAGE_SINGLE</td>
</tr>
<tr>
<td>6 EMCS_LOGLIKE_IMPROVEMENT</td>
<td>.001</td>
</tr>
<tr>
<td>7 EMCS_MAX_NUM_ATTR_2D</td>
<td>50</td>
</tr>
<tr>
<td>8 EMCS_MIN_PCT_ATTR_SUPPORT</td>
<td>.1</td>
</tr>
<tr>
<td>9 EMCS_MODEL_SEARCH</td>
<td>EMCS_MODEL_SEARCH_DISABLE</td>
</tr>
<tr>
<td>10 EMCS_NUM_COMPONENTS</td>
<td>20</td>
</tr>
<tr>
<td>11 EMCS_NUM_DISTRIBUTION</td>
<td>EMCS_NUM_DISTR_SYSTEM</td>
</tr>
<tr>
<td>12 EMCS_NUM_EQUIWIDTH_BINS</td>
<td>11</td>
</tr>
<tr>
<td>13 EMCS_NUM_ITERATIONS</td>
<td>100</td>
</tr>
<tr>
<td>14 EMCS_NUM_PROJECTIONS</td>
<td>50</td>
</tr>
<tr>
<td>15 EMCS_RANDOM_SEED</td>
<td>5</td>
</tr>
<tr>
<td>16 EMCS_REMOVE_COMPONENTS</td>
<td>EMCS_REMOVE_COMPS_ENABLE</td>
</tr>
<tr>
<td>17 ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>18 ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>19 ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>20 PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 EMCS_ATTRIBUTE_FILTER</td>
<td>EMCS_ATTR_FILTER_DISABLE</td>
</tr>
<tr>
<td>1 EMCS_CONVERGENCE_CRITERION</td>
<td>EMCS_CONV_CRIT_BIC</td>
</tr>
<tr>
<td>2 EMCS_NUM_QUANTILE_BINS</td>
<td>3</td>
</tr>
<tr>
<td>3 EMCS_NUM_TOPN_BINS</td>
<td>3</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CONVERGED</td>
<td>YES</td>
</tr>
<tr>
<td>1 LOGLIKELIHOOD</td>
<td>-1.75777</td>
</tr>
<tr>
<td>2 NUM_CLUSTERS</td>
<td>2</td>
</tr>
<tr>
<td>3 NUM_COMPONENTS</td>
<td>9</td>
</tr>
<tr>
<td>4 NUM_ROWS</td>
<td>104</td>
</tr>
<tr>
<td>5 RANDOM_SEED</td>
<td>5</td>
</tr>
<tr>
<td>6 REMOVED_COMPONENTS</td>
<td>11</td>
</tr>
</tbody>
</table>

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:
### Clusters

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>CLUSTER_NAME</th>
<th>LEFT_CHILD_ID</th>
<th>RECORD_COUNT</th>
<th>PARENT</th>
<th>TREE_LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>104</td>
<td>NaN</td>
<td>1</td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td>104</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>2</td>
<td>36</td>
<td>1.0</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
<td>68</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RIGHT_CHILD_ID</th>
</tr>
</thead>
</table>
| 0              | 3.0
| 1              | NaN
| 2              | NaN

### Taxonomy

<table>
<thead>
<tr>
<th>PARENT_CLUSTER_ID</th>
<th>CHILD_CLUSTER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
</tr>
</tbody>
</table>

### Centroids

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>MEAN</th>
<th>MODE_VALUE</th>
<th>VARIANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Length</td>
<td>3.721154</td>
<td>None</td>
<td>3.234694</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Width</td>
<td>1.155769</td>
<td>None</td>
<td>0.567539</td>
</tr>
<tr>
<td>2</td>
<td>Sepal_Length</td>
<td>5.831731</td>
<td>None</td>
<td>0.753255</td>
</tr>
<tr>
<td>3</td>
<td>Sepal_Width</td>
<td>3.074038</td>
<td>None</td>
<td>0.221358</td>
</tr>
<tr>
<td>4</td>
<td>Species</td>
<td>NaN</td>
<td>setosa</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>Petal_Length</td>
<td>1.488889</td>
<td>None</td>
<td>0.033016</td>
</tr>
<tr>
<td>6</td>
<td>Petal_Width</td>
<td>0.250000</td>
<td>None</td>
<td>0.012857</td>
</tr>
<tr>
<td>7</td>
<td>Sepal_Length</td>
<td>5.011111</td>
<td>None</td>
<td>0.113016</td>
</tr>
<tr>
<td>8</td>
<td>Sepal_Width</td>
<td>3.488889</td>
<td>None</td>
<td>0.134159</td>
</tr>
<tr>
<td>9</td>
<td>Species</td>
<td>NaN</td>
<td>setosa</td>
<td>NaN</td>
</tr>
<tr>
<td>10</td>
<td>Petal_Length</td>
<td>4.902941</td>
<td>None</td>
<td>0.860588</td>
</tr>
<tr>
<td>11</td>
<td>Petal_Width</td>
<td>1.635294</td>
<td>None</td>
<td>0.191572</td>
</tr>
<tr>
<td>12</td>
<td>Sepal_Length</td>
<td>6.266176</td>
<td>None</td>
<td>0.545555</td>
</tr>
<tr>
<td>13</td>
<td>Sepal_Width</td>
<td>2.854412</td>
<td>None</td>
<td>0.128786</td>
</tr>
<tr>
<td>14</td>
<td>Species</td>
<td>NaN</td>
<td>versicolor</td>
<td>NaN</td>
</tr>
</tbody>
</table>

### Leaf Cluster Counts

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>1</td>
<td>68</td>
</tr>
</tbody>
</table>

### Attribute Importance

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_IMPORTANCE_VALUE</th>
<th>ATTRIBUTE_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petal_Length</td>
<td>0.558311</td>
<td>2</td>
</tr>
<tr>
<td>Petal_Width</td>
<td>0.556300</td>
<td>3</td>
</tr>
<tr>
<td>Sepal_Length</td>
<td>0.469978</td>
<td>4</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td>0.196211</td>
<td>5</td>
</tr>
<tr>
<td>COMPONENT_ID</td>
<td>CLUSTER_ID</td>
<td>PRIOR_PROBABILITY</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Cluster Hists:

<table>
<thead>
<tr>
<th>cluster.id</th>
<th>variable</th>
<th>bin.id</th>
<th>lower.bound</th>
<th>upper.bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Length</td>
<td>1</td>
<td>1.00</td>
<td>1.59</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Length</td>
<td>2</td>
<td>1.59</td>
<td>2.18</td>
</tr>
<tr>
<td>2</td>
<td>Petal_Length</td>
<td>3</td>
<td>2.18</td>
<td>2.77</td>
</tr>
<tr>
<td>3</td>
<td>Petal_Length</td>
<td>4</td>
<td>2.77</td>
<td>3.36</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>137</td>
<td>Sepal_Width</td>
<td>11</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>138</td>
<td>Species:'Other'</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>139</td>
<td>Species:versicolor</td>
<td>3</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>140</td>
<td>Species:versicolor</td>
<td>2</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Label count:

<table>
<thead>
<tr>
<th>label</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>0</td>
</tr>
<tr>
<td>138</td>
<td>33</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>35</td>
</tr>
</tbody>
</table>

Rules:

<table>
<thead>
<tr>
<th>cluster.id</th>
<th>rhs.support</th>
<th>rhs.conf</th>
<th>lhr.support</th>
<th>lhs.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>104</td>
<td>1.00</td>
<td>93</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td>1</td>
<td>104</td>
<td>1.00</td>
<td>93</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td>1</td>
<td>104</td>
<td>1.00</td>
<td>93</td>
</tr>
<tr>
<td>Petal_Length</td>
<td>1</td>
<td>104</td>
<td>1.00</td>
<td>99</td>
</tr>
<tr>
<td>Petal_Length</td>
<td>1</td>
<td>104</td>
<td>1.00</td>
<td>99</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Explicit Semantic Analysis

The `oml.esa` class extracts text-based features from a corpus of documents and performs document similarity comparisons.

Explicit Semantic Analysis (ESA) is an unsupervised algorithm for feature extraction. ESA does not discover latent features but instead uses explicit features based on an existing knowledge base.

Explicit knowledge often exists in text form. Multiple knowledge bases are available as collections of text documents. These knowledge bases can be generic, such as Wikipedia, or domain-specific. Data preparation transforms the text into vectors that capture attribute-concept associations.

ESA uses concepts of an existing knowledge base as features rather than latent features derived by latent semantic analysis methods such as Singular Value Decomposition and Latent Dirichlet Allocation. Each row, for example, in a document in the training data maps to a feature, that is, a concept. ESA has multiple applications in the area of text processing, most notably semantic relatedness (similarity) and explicit topic modeling. Text similarity use cases might involve, for example, resume matching, searching for similar blog postings, and so on.

For information on the `oml.esa` class attributes and methods, invoke `help(oml.esa)` or see Oracle Machine Learning for Python API Reference.

Settings for an Explicit Semantic Analysis Model

The following table lists settings for ESA models.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESAS_MIN_ITEMS</td>
<td>A non-negative number</td>
<td>Determines the minimum number of non-zero entries required in an input row. The default value is 100 for text input and 0 for non-text input.</td>
</tr>
<tr>
<td>ESAS_TOPN_FEATURES</td>
<td>A positive integer</td>
<td>Controls the maximum number of features per attribute. The default value is 1000.</td>
</tr>
</tbody>
</table>
Table 5-8  (Cont.) Explicit Semantic Analysis Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESAS_VALUE_THRESHOLD</td>
<td>A non-negative number</td>
<td>Sets the threshold to a small value for attribute weights in the transformed build data. The default value is 1e-8.</td>
</tr>
<tr>
<td>FEAT_NUM_FEATURES</td>
<td>TO_CHAR(numeric_expr &gt;=1)</td>
<td>The number of features to extract. The default value is estimated by the algorithm. If the matrix rank is smaller than this number, then fewer features are returned.</td>
</tr>
</tbody>
</table>

See Also:
- About Model Settings
- Shared Settings

Example 5-11  Using the oml.esa Class

This example creates an ESA model and uses some of the methods of the oml.esa class.

```python
import oml
from oml import cursor
import pandas as pd

# Create training data and test data.
dat = oml.push(pd.DataFrame(
    {'COMMENTS':['Aids in Africa: Planning for a long war',
                 'Mars rover maneuvers for rim shot',
                 'Mars express confirms presence of water at Mars south pole',
                 'NASA announces major Mars rover finding',
                 'Drug access, Asia threat in focus at AIDS summit',
                 'NASA Mars Odyssey THEMIS image: typical crater',
                 'Road blocks for AIDS'],
    'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed = 1234)
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
cur = cursor()
cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()

odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
                 'ODMS_TEXT_MIN_DOCUMENTS': 1,
                 'ESAS_MIN_ITEMS': 1}
```
ctx_settings = {'COMMENTS': '
  TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)(TOKEN_TYPE:STEM)'}

# Create an oml ESA model object.
esa_mod = oml.esa(**odm_settings)

# Fit the ESA model according to the training data and parameter settings.
esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
  ctx_settings = ctx_settings)

# Show model details.
esa_mod

# Use the model to make predictions on test data.
esa_mod.predict(test_dat,
  supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])

esa_mod.transform(test_dat,
  supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
  topN = 2).sort_values(by = ['ID'])

esa_mod.feature_compare(test_dat,
  compare_cols = 'COMMENTS',
  supplemental_cols = ['ID'])

esa_mod.feature_compare(test_dat,
  compare_cols = ['COMMENTS', 'YEAR'],
  supplemental_cols = ['ID'])

# Change the setting parameter and refit the model.
new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
  'ODMS_TEXT_MAXFEATURES': '2',
  'ESAS_TOPNFEATURES': '2'}
esa_mod.set_params(**new_setting).fit(train_dat, 'ID', case_id = 'ID',
  ctx_settings = ctx_settings)

cur = cursor()
cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()

Listing for This Example

>>> import oml
>>> from oml import cursor
>>> import pandas as pd

>>> # Create training data and test data.
... dat = oml.push(pd.DataFrame(
...   {'COMMENTS': ['Aids in Africa: Planning for a long war',
...     'Mars rover maneuvers for rim shot',
...     'Mars express confirms presence of water at Mars south pole',
...     'NASA announces major Mars rover finding',
...     'Drug access, Asia threat in focus at AIDS summit',
...     'NASA Mars Odyssey THEMIS image: typical crater',
...     ...]})
...   ...)}
...   ...}
... 'Road blocks for Aids'},
... 'YEAR': ['2017', '2018', '2017', '2017', '2018', '2018'],
... 'lD': [1, 2, 3, 4, 5, 6, 7]).split(ratio=(0.7, 0.3), seed = 1234)
>>> train_dat = dat[0]
>>> test_dat = dat[1]

>>> # Specify settings.
... cur = cursor()
>>> cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY');
End;")
>>> cur.close()

>>> odm_settings = {"odms_text_policy_name": "DMDEMO_ESA_POLICY",
... "ODMS_TEXT_MIN_DOCUMENTS": 1,
... "ESAS_MIN_ITEMS": 1}

>>> ctx_settings = {'COMMENTS':
... "TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)
... (TOKEN_TYPE:STEM)'

>>> # Create an oml ESA model object.
... esa_mod = oml.esa(**odm_settings)

>>> # Fit the ESA model according to the training data and parameter
settings.
... esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
... ctx_settings = ctx_settings)

>>> # Show model details.
... esa_mod

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_EXPLICIT_SEMANTIC_ANALYSIS</td>
</tr>
<tr>
<td>ESAS_MIN_ITEMS</td>
<td>1</td>
</tr>
<tr>
<td>ESAS_TOPN_FEATURES</td>
<td>1000</td>
</tr>
<tr>
<td>ESAS_VALUE_THRESHOLD</td>
<td>.00000001</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>ODMS_TEXT_MAX_FEATURES</td>
<td>300000</td>
</tr>
<tr>
<td>ODMS_TEXT_MIN_DOCUMENTS</td>
<td>1</td>
</tr>
<tr>
<td>ODMS_TEXT_POLICY_NAME</td>
<td>DMDEMO_ESA_POLICY</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:
<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_ROWS</td>
<td>4</td>
</tr>
</tbody>
</table>

Attributes:
Comments

YEAR

Partition: NO

Features:

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>COMMENTS.AFRICA</td>
<td>None</td>
<td>0.342997</td>
</tr>
<tr>
<td>1</td>
<td>COMMENTS.AIDS</td>
<td>None</td>
<td>0.171499</td>
</tr>
<tr>
<td>2</td>
<td>COMMENTS.LONG</td>
<td>None</td>
<td>0.342997</td>
</tr>
<tr>
<td>3</td>
<td>COMMENTS.PLANNING</td>
<td>None</td>
<td>0.342997</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>24</td>
<td>COMMENTS.ODYSSEY</td>
<td>None</td>
<td>0.282843</td>
</tr>
<tr>
<td>25</td>
<td>COMMENTS.THEMIS</td>
<td>None</td>
<td>0.282843</td>
</tr>
<tr>
<td>26</td>
<td>COMMENTS.TYPICAL</td>
<td>None</td>
<td>0.282843</td>
</tr>
<tr>
<td>27</td>
<td>YEAR</td>
<td>2018</td>
<td>0.707107</td>
</tr>
</tbody>
</table>

>>> # Use the model to make predictions on test data.
... esa_mod.predict(test_dat,
...                 supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])

ID        COMMENTS  FEATURE_ID
0          NASA announces major Mars rover finding 3
1          NASA Mars Odyssey THEMIS image: typical crater 2
2          Road blocks for Aids 5

>>> esa_mod.transform(test_dat,
...                      supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])

COMMENTS  TOP_1  TOP_1_VAL  \
0          NASA announces major Mars rover finding 3 0.647065
1          NASA Mars Odyssey THEMIS image: typical crater 2 0.766237
2          Road blocks for Aids 5 0.759125

TOP_2  TOP_2_VAL
0  1  0.590565
1  2  0.616672
2  2  0.632604

>>> esa_mod.feature_compare(test_dat,
...                         compare_cols = ['COMMENTS'],
...                         supplemental_cols = ['ID'])

ID_A  ID_B  SIMILARITY
0     4     6   0.946469
1     4     7   0.871994
2     6     7   0.954565

>>> esa_mod.feature_compare(test_dat,
...                         compare_cols = ['COMMENTS', 'YEAR'],
...                         supplemental_cols = ['ID'])

ID_A  ID_B  SIMILARITY
0     4     6   0.467644
1     4     7   0.377144
>>> # Change the setting parameter and refit the model.
>>> new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
...                'ODMS_TEXT_MAX_FEATURES': '2',
...                'ESAS_TOPN_FEATURES': '2'}
>>> esa_mod.set_params(**new_setting).fit(train_dat, case_id = 'ID',
...                                       ctx_settings = ctx_settings)

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_EXPLICIT_SEMANTIC_ANALYS</td>
</tr>
<tr>
<td>ESAS_MIN_ITEMS</td>
<td>1</td>
</tr>
<tr>
<td>ESAS_TOPN_FEATURES</td>
<td>2</td>
</tr>
<tr>
<td>ESAS_VALUE_THRESHOLD</td>
<td>0.01</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>ODMS_TEXT_MAX_FEATURES</td>
<td>2</td>
</tr>
<tr>
<td>ODMS_TEXT_MINDOCUMENTS</td>
<td>1</td>
</tr>
<tr>
<td>ODMS_POLICY_NAME</td>
<td>DMDEMO_ESA_POLICY</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_ROWS</td>
<td>4</td>
</tr>
</tbody>
</table>

Attributes:

- COMMENTS
- YEAR

Partition: NO

Features:

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COMMENTS.AIDS</td>
<td>None</td>
<td>0.707107</td>
</tr>
<tr>
<td>2</td>
<td>YEAR</td>
<td>2017</td>
<td>0.707107</td>
</tr>
<tr>
<td>3</td>
<td>COMMENTS.MARS</td>
<td>None</td>
<td>0.707107</td>
</tr>
<tr>
<td>4</td>
<td>YEAR</td>
<td>2018</td>
<td>0.707107</td>
</tr>
<tr>
<td>5</td>
<td>COMMENTS.MARS</td>
<td>None</td>
<td>0.707107</td>
</tr>
<tr>
<td>7</td>
<td>YEAR</td>
<td>2018</td>
<td>0.707107</td>
</tr>
</tbody>
</table>

>>> cur = cursor()
>>> cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
Generalized Linear Model

The `oml.glm` class builds a Generalized Linear Model (GLM) model.

GLM models include and extend the class of linear models. They relax the restrictions on linear models, which are often violated in practice. For example, binary (yes/no or 0/1) responses do not have the same variance across classes.

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.

In addition to the classical weighted least squares estimation for linear regression and iteratively re-weighted least squares estimation for logistic regression, both solved through Cholesky decomposition and matrix inversion, Oracle Machine Learning GLM provides a conjugate gradient-based optimization algorithm that does not require matrix inversion and is very well suited to high-dimensional data. The choice of algorithm is handled internally and is transparent to the user.

GLM can be used to build classification or regression models as follows:

- **Classification**: Binary logistic regression is the GLM classification algorithm. The algorithm uses the logit link function and the binomial variance function.
- **Regression**: Linear regression is the GLM regression algorithm. The algorithm assumes no target transformation and constant variance over the range of target values.

The `oml.glm` class allows you to build two different types of models. Some arguments apply to classification models only and some to regression models only.

For information on the `oml.glm` class attributes and methods, invoke `help(oml.glm)` or see *Oracle Machine Learning for Python API Reference*.

**Settings for a Generalized Linear Model**

The following table lists the settings that apply to GLM models.

### Table 5-9 Generalized Linear Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAS_COST_TABLE_NAME</td>
<td><code>table_name</code></td>
<td>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table.</td>
</tr>
</tbody>
</table>
|                        |               | - Column Name: ACTUAL_TARGET_VALUE  
|                        |               |   Data Type: Valid target data type  
|                        |               | - Column Name: PREDICTED_TARGET_VALUE  
|                        |               |   Data Type: Valid target data type  
|                        |               | - Column Name: COST  
<p>|                        |               |   Data Type: NUMBER  |</p>
<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>ON OFF</td>
<td>Indicates whether 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.</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_TABLE_NAME</td>
<td>table_name</td>
<td>The name of a table that stores weighting information for individual target values in 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. The following are the column requirements for the table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column Name: TARGET_VALUE Data Type: Valid target data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column Name: CLASS_WEIGHT Data Type: NUMBER</td>
</tr>
<tr>
<td>GLMS_BATCH_ROWS</td>
<td>0 or a positive integer.</td>
<td>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 value is 2000.</td>
</tr>
<tr>
<td>GLMS_CONF_LEVEL</td>
<td>TO_CHAR(0&lt; numeric_expr &lt;1)</td>
<td>The confidence level for coefficient confidence intervals. The default confidence level is 0.95.</td>
</tr>
<tr>
<td>GLMS_CONV_TOLERANCE</td>
<td>The range is (0, 1) non-inclusive.</td>
<td>Convergence tolerance setting of the GLM algorithm. The default value is system-determined.</td>
</tr>
<tr>
<td>GLMS_FTR_GEN_METHOD</td>
<td>GLMS_FTR_GEN_CUBIC GLMS_FTR_GEN_QUADRIC</td>
<td>Whether feature generation is cubic or quadratic. When you enable feature generation, the algorithm automatically chooses the most appropriate feature generation method based on the data.</td>
</tr>
<tr>
<td>GLMS_FTR_GENERATION</td>
<td>GLMS_FTR_GENERATION_ENABLE GLMS_FTR_GENERATION_DISABLE</td>
<td>Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled.</td>
</tr>
<tr>
<td>GLMS_FTR_SEL_CRIT</td>
<td>GLMS_FTR_SEL_AIC GLMS_FTR_SEL_ALPHA_INV GLMS_FTR_SEL_RIC GLMS_FTR_SEL_SBIC</td>
<td>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.</td>
</tr>
<tr>
<td>Note: Feature generation can only be enabled when feature selection is also enabled.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:

- Feature generation can only be enabled when feature selection is also enabled.
### Table 5-9 (Cont.) Generalized Linear Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLMS_FTR_SELECTION</td>
<td>GLMS_FTR_SELECTION_DISABLE</td>
<td>Enable or disable feature selection for GLM. By default, feature selection is not enabled.</td>
</tr>
<tr>
<td>GLMS_MAX_FEATURES</td>
<td>TO_CHAR(0 &lt; numeric_expr &lt;= 2000)</td>
<td>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.</td>
</tr>
<tr>
<td>GLMS_NUM_ITERATIONS</td>
<td>A positive integer.</td>
<td>Maximum number of iterations for the GLM algorithm. The default value is system-determined.</td>
</tr>
<tr>
<td>GLMS_PRUNE_MODEL</td>
<td>GLMS_PRUNE_MODEL_ENABLE</td>
<td>When feature selection is enabled, the algorithm automatically performs pruning based on the data.</td>
</tr>
<tr>
<td>GLMS_REFERENCE_CLASS_NAME</td>
<td>target_value</td>
<td>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.</td>
</tr>
<tr>
<td>GLMS_RIDGE_REGRESSION</td>
<td>GLMS_RIDGE_REG_ENABLE</td>
<td>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.</td>
</tr>
<tr>
<td>GLMS_RIDGE_VALUE</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>The value of the ridge parameter. Use this setting only when you have configured the algorithm to use ridge regression. If ridge regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm.</td>
</tr>
<tr>
<td>GLMS_ROW_DIAGNOSTICS</td>
<td>GLMS_ROW_DIAG_ENABLE</td>
<td>Enable or disable row diagnostics. By default, row diagnostics are disabled.</td>
</tr>
<tr>
<td>GLMS_SOLVER</td>
<td>GLMS_SOLVER_CHOL</td>
<td>Specifies the GLM solver. You cannot select the solver if GLMS_FTR_SELECTION setting is enabled. The default value is system determined. The GLMS_SOLVER_CHOL solver uses Cholesky decomposition. The GLMS_SOLVER_SGD solver uses stochastic gradient descent.</td>
</tr>
<tr>
<td>GLMS_SPARSE_SOLVER</td>
<td>GLMS_SPARSE_SOLVER_ENABLE</td>
<td>Enable or disable the use of a sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.</td>
</tr>
</tbody>
</table>
### Table 5-9  (Cont.) Generalized Linear Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMS_ROW_WEIGHT_COLUMN_NAME</td>
<td>column_name</td>
<td>The name of a column in the training data that contains a weighting factor for the rows. The column datatype must be NUMBER.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You can use row weights as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. You can also use row weights 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.</td>
</tr>
</tbody>
</table>

### See Also:

- About Model Settings
- Shared Settings

### Example 5-12  Using the oml.glm Class

This example demonstrates the use of various methods of the oml.glm class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Petal_Width')
train_y = dat[0]['Petal_Width']
```
test_dat = dat[1]

# Specify settings.
setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}

# Create a GLM model object.
glm_mod = oml.glm("regression", **setting)

# Fit the GLM model according to the training data and parameter settings.
glm_mod = glm_mod.fit(train_x, train_y)

# Show the model details.
glm_mod

# Use the model to make predictions on the test data.
glm_mod.predict(test_dat.drop('Petal_Width'),
                 supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Petal_Length', 'Species']])

# Return the prediction probability.
glm_mod.predict(test_dat.drop('Petal_Width'),
                 supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Petal_Length', 'Species']],
                 proba = True)

glm_mod.score(test_dat.drop('Petal_Width'),
              test_dat[:, ['Petal_Width']])

# Change the parameter setting and refit the model.
new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
glm_mod.set_params(**new_setting).fit(train_x, train_y)

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
... x = pd.DataFrame(iris.data,
...                   columns = ['Sepal_Length', 'Sepal_Width',
...                              'Petal_Length', 'Petal_Width'])

>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                   columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
>>> # Create the IRIS database table and the proxy object for the table.
>>> oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
>>> dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Petal_Width')
>>> train_y = dat[0]['Petal_Width']
>>> test_dat = dat[1]

>>> # Specify settings.
>>> setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}

>>> # Create a GLM model object.
>>> glm_mod = oml.glm("regression", **setting)

>>> # Fit the GLM model according to the training data and parameter settings.
>>> glm_mod = glm_mod.fit(train_x, train_y)

>>> # Show the model details.
>>> glm_mod

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ALGO_NAME</td>
<td>ALGO_GENERALIZED_LINEAR_MODEL</td>
</tr>
<tr>
<td>1 GLMS_CONF_LEVEL</td>
<td>.95</td>
</tr>
<tr>
<td>2 GLMS_FTR_GENERATION</td>
<td>GLMS_FTR_GENERATION_DISABLE</td>
</tr>
<tr>
<td>3 GLMS_FTR_SELECTION</td>
<td>GLMS_FTR_SELECTION_DISABLE</td>
</tr>
<tr>
<td>4 GLMS_SOLVER</td>
<td>GLMS_SOLVER_QR</td>
</tr>
<tr>
<td>5 ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>6 ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>7 ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>8 PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 GLMS_CONV_TOLERANCE</td>
<td>.000000500000000000000004</td>
</tr>
<tr>
<td>1 GLMS_NUM_ITERATIONS</td>
<td>30</td>
</tr>
<tr>
<td>2 GLMS_RIDGE_REGRESSION</td>
<td>GLMS_RIDGE_REG_ENABLE</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ADJUSTED_R_SQUARE</td>
<td>0.949634</td>
</tr>
<tr>
<td>1 AIC</td>
<td>-363.888</td>
</tr>
<tr>
<td>2 COEFF_VAR</td>
<td>14.6284</td>
</tr>
<tr>
<td>3 CONVERGED</td>
<td>YES</td>
</tr>
<tr>
<td>4 CORRECTED_TOTAL_DF</td>
<td>103</td>
</tr>
</tbody>
</table>
5  CORRECTED_TOT_SS    58.4565
6  DEPENDENT_MEAN    1.15577
7  ERROR_DF    98
8  ERROR_MEAN_SQUARE    0.028585
9  ERROR_SUM_SQUARES    2.80131
10  F_VALUE    389.405
11  GMSEP    0.030347
12  HOCKING_SP    0.000295
13  J_P    0.030234
14  MODEL_DF    5
15  MODEL_F_P_VALUE    0
16  MODEL_MEAN_SQUARE    11.131
17  MODEL_SUM_SQUARES    55.6552
18  NUM_PARAMS    6
19  NUM_ROWS    104
20  RANK_DEFICIENCY    0
21  ROOT_MEAN_SQ    0.16907
22  R_SQ    0.952079
23  SBIC    -348.021
24  VALID_COVARIANCE_MATRIX    YES

[1 rows x 25 columns]

Attributes:
Petal_Length
Sepal_Length
Sepal_Width
Species
Partition: NO

Coefficients:

name       level  estimate
0   (Intercept)        None -0.600603
1  Petal_Length        None  0.239775
2  Sepal_Length        None -0.078338
3   Sepal_Width        None  0.253996
4       Species  versicolor  0.652420
5       Species   virginica  1.010438

Fit Details:

name         value
0         ADJUSTED_R_SQUARE  9.496338e-01
1                       AIC -3.638876e+02
2                 COEFF_VAR  1.462838e+01
3        CORRECTED_TOTAL_DF  1.030000e+02
... 21              ROOT_MEAN_SQ  1.690704e-01
22                  R_SQ  9.520788e-01
23                     SBIC -3.480213e+02
24  VALID_COVARIANCE_MATRIX  1.000000e+00

Rank:
6

Chapter 5
Generalized Linear Model
Deviance:
2.801309

AIC:
-364

Null Deviance:
58.456538

DF Residual:
98.0

DF Null:
103.0

Converged:
True

```python
>>> # Use the model to make predictions on the test data.
... glm_mod.predict(test_dat.drop('Petal_Width'),
...                 supplemental_cols = test_dat[:,
...                    ['Sepal_Length', 'Sepal_Width',
...                    'Petal_Length', 'Species'])
                  Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
     0            4.9          3.0           1.4      setosa    0.113215
     1            4.9          3.1           1.5      setosa    0.162592
     2            4.8          3.4           1.6      setosa    0.270602
     3            5.8          4.0           1.2      setosa    0.248752
     ...          ...          ...           ...         ...         ...
     42           6.7          3.3           5.7   virginica    2.89876
     43           6.7          3.0           5.2   virginica    1.893790
     44           6.5          3.0           5.2   virginica    1.909457
     45           5.9          3.0           5.1   virginica    1.932483

>>> # Return the prediction probability.
... glm_mod.predict(test_dat.drop('Petal_Width'),
...                 supplemental_cols = test_dat[:,
...                    ['Sepal_Length', 'Sepal_Width',
...                    'Petal_Length', 'Species'])
    proba = True)
                  Sepal_Length  Sepal_Width  Species  PREDICTION
     0            4.9          3.0      setosa    0.113215
     1            4.9          3.1      setosa    0.162592
     2            4.8          3.4      setosa    0.270602
     3            5.8          4.0      setosa    0.248752
     ...          ...          ...         ...         ...
     42           6.7          3.3   virginica    2.089876
Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_GENERALIZED_LINEAR_MODEL</td>
</tr>
<tr>
<td>GLMS_CONF_LEVEL</td>
<td>.95</td>
</tr>
<tr>
<td>GLMS_FTR_GENERATION</td>
<td>GLMS_FTR_GENERATION_DISABLE</td>
</tr>
<tr>
<td>GLMS_FTR_SELECTION</td>
<td>GLMS_FTR_SELECTION_DISABLE</td>
</tr>
<tr>
<td>GLMS_SOLVER</td>
<td>GLMS_SOLVER_SGD</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLMS_BATCH_ROWS</td>
<td>2000</td>
</tr>
<tr>
<td>GLMS_CONV_TOLERANCE</td>
<td>.0001</td>
</tr>
<tr>
<td>GLMS_NUM_ITERATIONS</td>
<td>500</td>
</tr>
<tr>
<td>GLMS_RIDGE_REGRESSION</td>
<td>GLMS_RIDGE_REG_ENABLE</td>
</tr>
<tr>
<td>GLMS_RIDGE_VALUE</td>
<td>.01</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUSTED_R_SQUARE</td>
<td>0.94175</td>
</tr>
<tr>
<td>AIC</td>
<td>-348.764</td>
</tr>
<tr>
<td>COEFF_VAR</td>
<td>15.7316</td>
</tr>
<tr>
<td>CONVERGED</td>
<td>NO</td>
</tr>
<tr>
<td>CORRECTED_TOTAL_DF</td>
<td>103</td>
</tr>
<tr>
<td>CORRECTED_TOT_SS</td>
<td>58.4565</td>
</tr>
<tr>
<td>DEPENDENT_MEAN</td>
<td>1.15577</td>
</tr>
<tr>
<td>ERROR_DF</td>
<td>98</td>
</tr>
<tr>
<td>ERROR_MEAN_SQUARE</td>
<td>0.033059</td>
</tr>
<tr>
<td>ERROR_SUM_SQUARES</td>
<td>3.23979</td>
</tr>
<tr>
<td>F_VALUE</td>
<td>324.347</td>
</tr>
<tr>
<td>GMSEP</td>
<td>0.035097</td>
</tr>
<tr>
<td>HOCKING_SP</td>
<td>0.000341</td>
</tr>
<tr>
<td>J_P</td>
<td>0.034966</td>
</tr>
<tr>
<td>MODEL_DF</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>MODEL_F_P_VALUE</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
</tr>
<tr>
<td>15</td>
<td>MODEL_MEAN_SQUARE</td>
</tr>
<tr>
<td>16</td>
<td>MODEL_SUM_SQUARES</td>
</tr>
<tr>
<td>17</td>
<td>NUM_PARAMS</td>
</tr>
<tr>
<td>18</td>
<td>NUM_ROWS</td>
</tr>
<tr>
<td>19</td>
<td>RANK_DEFICIENCY</td>
</tr>
<tr>
<td>20</td>
<td>ROOT_MEAN_SQ</td>
</tr>
<tr>
<td>21</td>
<td>R_SQ</td>
</tr>
<tr>
<td>22</td>
<td>SBIC</td>
</tr>
<tr>
<td>23</td>
<td>VALID_COVARIANCE_MATRIX</td>
</tr>
</tbody>
</table>

[1 rows x 25 columns]

Attributes:
- Petal_Length
- Sepal_Length
- Sepal_Width
- Species

Partition: NO

Coefficients:

<table>
<thead>
<tr>
<th>name</th>
<th>level</th>
<th>estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>None</td>
<td>-0.338046</td>
</tr>
<tr>
<td>Petal_Length</td>
<td>None</td>
<td>0.378658</td>
</tr>
<tr>
<td>Sepal_Length</td>
<td>None</td>
<td>-0.084440</td>
</tr>
<tr>
<td>Sepal_Width</td>
<td>None</td>
<td>0.137150</td>
</tr>
<tr>
<td>Species</td>
<td>versicolor</td>
<td>0.151916</td>
</tr>
<tr>
<td>Species</td>
<td>virginica</td>
<td>0.337535</td>
</tr>
</tbody>
</table>

Fit Details:

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUSTED_R_SQUARE</td>
<td>9.417502e-01</td>
</tr>
<tr>
<td>AIC</td>
<td>-3.487639e+02</td>
</tr>
<tr>
<td>COEFF_VAR</td>
<td>1.573164e+01</td>
</tr>
<tr>
<td>CORRECTED_TOTAL_DF</td>
<td>1.030000e+02</td>
</tr>
<tr>
<td>ROOT_MEAN_SQ</td>
<td>1.818215e-01</td>
</tr>
<tr>
<td>R_SQ</td>
<td>9.445778e-01</td>
</tr>
<tr>
<td>SBIC</td>
<td>-3.328975e+02</td>
</tr>
</tbody>
</table>

Rank:

6

Deviance:

3.239787

AIC:

-349
Null Deviance:
58.456538

Prior Weights:
1

DF Residual:
98.0

DF Null:
103.0

Converged:
False

**k-Means**

The `oml.km` class uses the k-Means (KM) algorithm, which is a hierarchical, distance-based clustering algorithm that partitions data into a specified number of clusters.

The algorithm has the following features:

- Several distance functions: Euclidean, Cosine, and Fast Cosine distance functions. The default is Euclidean.
- 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 numeric attributes.

For information on the `oml.km` class attributes and methods, invoke `help(oml.km)` or see Oracle Machine Learning for Python API Reference.

**Settings for a k-Means Model**

The following table lists the settings that apply to KM models.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUS_NUM_CLUSTERS</td>
<td><code>TO_CHAR(numeric_expr &gt;= 1)</code></td>
<td>The maximum number of leaf clusters generated by the algorithm. The algorithm produces the specified number of clusters unless there are fewer distinct data points. The default value is 10.</td>
</tr>
</tbody>
</table>
Table 5-10  (Cont.) k-Means Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMNS_CONV_TOLERANCE</td>
<td>TO_CHAR(0&lt; numeric_expr &lt;1)</td>
<td>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.</td>
</tr>
<tr>
<td>KMNS_DETAILS</td>
<td>KMNS_DETAILS_ALL</td>
<td>Determines the level of cluster detail that is computed during the build. KMNS_DETAILS_ALL: Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed. KMNS_DETAILS_HIERARCHY: Cluster hierarchy and cluster record counts are computed. This is the default value. KMNS_DETAILS_NONE: No cluster details are computed. Only the scoring information is persisted.</td>
</tr>
<tr>
<td>KMNS_DISTANCE</td>
<td>KMNS_COSINE</td>
<td>Distance function for k-Means. The default distance function is KMNS_EUCLIDEAN.</td>
</tr>
<tr>
<td>KMNS_ITERATIONS</td>
<td>TO_CHAR(positive_numeric_expr)</td>
<td>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.</td>
</tr>
<tr>
<td>KMNS_MIN_PCT_ATTR_SUP</td>
<td>TO_CHAR(0 &lt;= numeric_expr &lt;= 1)</td>
<td>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.</td>
</tr>
<tr>
<td>KMNS_NUM_BINS</td>
<td>TO_CHAR(numeric_expr &gt; 0)</td>
<td>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, which have only one bin. The default number of histogram bins is 11.</td>
</tr>
<tr>
<td>KMNS_RANDOM_SEED</td>
<td>Non-negative integer</td>
<td>Controls the seed of the random generator used during the k-Means initialization. It must be a non-negative integer value. The default value is 0.</td>
</tr>
</tbody>
</table>
### Table 5-10  (Cont.) k-Means Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMNS_SPLIT_CRITERION</td>
<td>KMNS_SIZE</td>
<td>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.</td>
</tr>
<tr>
<td>KMNS_VARIANCE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**See Also:**
- About Model Settings
- Shared Settings

### Example 5-13  Using the oml.km Class

This example creates a KM model and uses methods of it. In the listing for this example, some of the output is not shown as indicated by ellipses.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
    {0: 'setosa', 1: 'versicolor', 2:'virginica'}[x], iris.target)),
    columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
```
test_dat = dat[1]

# Specify settings.
setting = {'kmns_iterations': 20}

# Create a KM model object and fit it.
km_mod = oml.km(n_clusters = 3, **setting).fit(train_dat)

# Show model details.
km_mod

# Use the model to make predictions on the test data.
km_mod.predict(test_dat,
supplemental_cols =
    test_dat[:, ['Sepal_Length', 'Sepal_Width',
                'Petal_Length', 'Species']])
km_mod.predict_proba(test_dat,
supplemental_cols =
    test_dat[:, ['Species']]).sort_values(by =
    ['Species', 'PROBABILITY_OF_3'])
km_mod.transform(test_dat)
km_mod.score(test_dat)

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
... x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
... y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...   oml.drop('IRIS')
... except:
...   pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
... train_dat = dat[0]
... test_dat = dat[1]
>>> # Specify settings.
... setting = {'kmns_iterations': 20}

>>> # Create a KM model object and fit it.
... km_mod = omlkm(n_clusters = 3, **setting).fit(train_dat)

>>> # Show model details.
... km_mod

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_KMEANS</td>
</tr>
<tr>
<td>CLUS_NUM_CLUSTERS</td>
<td>3</td>
</tr>
<tr>
<td>KMNS_CONV_TOLERANCE</td>
<td>.001</td>
</tr>
<tr>
<td>KMNS_DETAILS</td>
<td>KMNS_DETAILS_HIERARCHY</td>
</tr>
<tr>
<td>KMNS_DISTANCE</td>
<td>KMNS_EUCLIDEAN</td>
</tr>
<tr>
<td>KMNS_ITERATIONS</td>
<td>20</td>
</tr>
<tr>
<td>KMNS_MIN_PCT_ATTR_SUPPORT</td>
<td>.1</td>
</tr>
<tr>
<td>KMNS_NUM_BINS</td>
<td>11</td>
</tr>
<tr>
<td>KMNS_RANDOM_SEED</td>
<td>0</td>
</tr>
<tr>
<td>KMNS_SPLIT_CRITERION</td>
<td>KMNS_VARIANCE</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERGED</td>
<td>YES</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>104.0</td>
</tr>
</tbody>
</table>

Attributes: Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>ROW_CNT</th>
<th>PARENT_CLUSTER_ID</th>
<th>TREE_LEVEL</th>
<th>DISPERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>NaN</td>
<td>1</td>
<td>0.986153</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.0</td>
<td>2</td>
<td>1.102147</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.0</td>
<td>2</td>
<td>0.767052</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2.0</td>
<td>3</td>
<td>1.015669</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2.0</td>
<td>3</td>
<td>1.205363</td>
</tr>
</tbody>
</table>

Taxonomy:
Leaf Cluster Counts:

<table>
<thead>
<tr>
<th>CLUSTER_ID</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

```python
>>> # Use the model to make predictions on the test data.
>>> km_mod.predict(test_dat, ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species'])
```

```python
Sepal_Length  Sepal_Width  Petal_Length     Species  CLUSTER_ID
0            4.9          3.0           1.4      setosa           3
1            4.9          3.1           1.5      setosa           3
2            4.8          3.4           1.6      setosa           3
3            5.8          4.0           1.2      setosa           3
```

```python
>>> km_mod.predict_proba(test_dat, supplemental_cols = test_dat[:, ['Species']]).sort_values(by = ['Species', 'PROBABILITY_OF_3'])
```

```python
Species  PROBABILITY_OF_3  PROBABILITY_OF_4  PROBABILITY_OF_5
0       setosa          0.791267          0.208494          0.000240
1       setosa          0.971498          0.028350          0.000152
2       setosa          0.981020          0.018499          0.000481
3       setosa          0.981907          0.017989          0.000104
```

```python
>>> km_mod.transform(test_dat)
```

```python
CLUSTER_DISTANCE
0           1.050234
1           0.859817
2           0.321065
```

### Chapter 5

#### k-Means
Naive Bayes

The oml.nb class creates a Naive Bayes (NB) model for classification.

The Naive Bayes algorithm is based on conditional probabilities. Naive Bayes looks at the historical data and calculates conditional probabilities for the target values by observing the frequency of attribute values and of combinations of attribute values.

Naive Bayes assumes that each predictor is conditionally independent of the others. (Bayes' Theorem requires that the predictors be independent.)

For information on the oml.nb class attributes and methods, invoke help(oml.nb) or see Oracle Machine Learning for Python API Reference.

Settings for a Naive Bayes Model

The following table lists the settings that apply to NB models.

Table 5-11  Naive Bayes Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLAS_COST_TABLE_NAME         | table_name    | The name of a table that stores a cost matrix for the algorithm to use in building the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table.  
  • Column Name: ACTUAL_TARGET_VALUE  
    Data Type: Valid target data type  
  • Column Name: PREDICTED_TARGET_VALUE  
    Data Type: Valid target data type  
  • Column Name: COST  
    Data Type: NUMBER |
| CLAS_MAX_SUP_BINS            | 2 <= a number <= 2147483647 | Specifies the maximum number of bins for each attribute.  
  The default value is 32. |
| CLAS_PRIORS_TABLE_NAME       | table_name    | The 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. The following are the column requirements for the table.  
  • Column Name: TARGET_VALUE  
    Data Type: Valid target data type  
  • Column Name: PRIOR_PROBABILITY  
    Data Type: NUMBER |
Table 5-11  (Cont.) Naive Bayes Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>ON OFF</td>
<td>Indicates whether 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.</td>
</tr>
<tr>
<td>NABS_PAIRWISE_THRESH</td>
<td>TO_CHAR(0 &lt;= numeric_expr &lt;= 1)</td>
<td>Value of the pairwise threshold for the NB algorithm. The default value is 0.</td>
</tr>
<tr>
<td>NABS_SINGLETON_THRESH</td>
<td>TO_CHAR(0 &lt;= numeric_expr &lt;= 1)</td>
<td>Value of the singleton threshold for the NB algorithm. The default value is 0.</td>
</tr>
</tbody>
</table>

See Also:
- About Model Settings
- Shared Settings

Example 5-14  Using the oml.nb Class

This example creates an NB model and uses some of the methods of the oml.nb class.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length', 'Sepal_Width',
                'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
    columns = ['Species'])

try:
    oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
```

Chapter 5  Naive Bayes

5-70
dat = oml.sync(table = 'IRIS').split()

train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# User specified settings.
setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}

# Create an oml NB model object.
nb_mod = oml.nb(**setting)

# Fit the NB model according to the training data and parameter
# settings.
nb_mod = nb_mod.fit(train_x, train_y)

# Show details of the model.
nb_mod

# Create a priors table in the database.
priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
priors = oml.create(pd.DataFrame(list(priors.items()),
columns=['TARGET_VALUE',
'PRIOR_PROBABILITY'],
table='NB_PRIOR_PROBABILITY_DEMO'))

# Change the setting parameter and refit the model
# with a user-defined prior table.
new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
nb_mod = nb_mod.set_params(**new_setting).fit(train_x, train_y,
                             priors=priors)

# Use the model to make predictions on test data.
nb_mod.predict(test_dat.drop('Species'),
                   supplemental_cols = test_dat[:, ['Sepal_Length',
                                                  'Sepal_Width',
                                                  'Petal_Length',
                                                  'Species']])

# Return the prediction probability.
nb_mod.predict(test_dat.drop('Species'),
                   supplemental_cols = test_dat[:, ['Sepal_Length',
                                                  'Sepal_Width',
                                                  'Species']],
                   proba = True)

# Return the top two most influencial attributes of the highest
# probability class.
nb_mod.predict(test_dat.drop('Species'),
                   supplemental_cols = test_dat[:, ['Sepal_Length',
                                                  'Sepal_Width',
                                                  'Petal_Length',
                                                  'Species']])
topNAttrs = 2)

# Make predictions and return the probability for each class
# on new data.
nb_mod.predict_proba(test_dat.drop('Species'),
    supplemental_cols = test_dat[:, ['Sepal_Length', 'Species']].sort_values(by = ['Sepal_Length', 'Species', 'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])

# Make predictions on new data and return the mean accuracy.
nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

**Listing for This Example**

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
>>> dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

>>> # User specified settings.
... setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}

>>> # Create an oml NB model object.
... nb_mod = oml.nb(**setting)

>>> # Fit the NB model according to the training data and parameter
```
... # settings.
>>> nb_mod = nb_mod.fit(train_x, train_y)

# Show details of the model.
... nb_mod

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME</td>
</tr>
<tr>
<td>1</td>
<td>CLAS_WEIGHTS_BALANCED</td>
</tr>
<tr>
<td>2</td>
<td>NABS_PAIRWISE_THRESHOLD</td>
</tr>
<tr>
<td>3</td>
<td>NABS_SINGLETON_THRESHOLD</td>
</tr>
<tr>
<td>4</td>
<td>ODMS_DETAILS</td>
</tr>
<tr>
<td>5</td>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
</tr>
<tr>
<td>6</td>
<td>ODMS_SAMPLING</td>
</tr>
<tr>
<td>7</td>
<td>PREP_AUTO</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUM_ROWS</td>
</tr>
</tbody>
</table>

Attributes:

- Petal_Length
- Petal_Width
- Sepal_Length
- Sepal_Width

Partition: NO

Priors:

<table>
<thead>
<tr>
<th>TARGET_NAME</th>
<th>TARGET_VALUE</th>
<th>PRIOR_PROBABILITY</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Species</td>
<td>setosa</td>
<td>0.333333</td>
</tr>
<tr>
<td>1</td>
<td>Species</td>
<td>versicolor</td>
<td>0.333333</td>
</tr>
<tr>
<td>2</td>
<td>Species</td>
<td>virginica</td>
<td>0.333333</td>
</tr>
</tbody>
</table>

Conditionals:

<table>
<thead>
<tr>
<th>TARGET_NAME</th>
<th>TARGET_VALUE</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
<th>ATTRIBUTE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>1.05</td>
</tr>
<tr>
<td>1</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>(1.05; 1.2]</td>
</tr>
<tr>
<td>2</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>(1.2; 1.35]</td>
</tr>
<tr>
<td>3</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>(1.35; 1.45]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>152</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>(3.25; 3.35]</td>
</tr>
<tr>
<td>153</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>(3.35; 3.45]</td>
</tr>
</tbody>
</table>
>>> # Create a priors table in the database.
... priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
... priors = oml.create(pd.DataFrame(list(priors.items()),
...                        columns = ['TARGET_VALUE',
...                                   'PRIOR_PROBABILITY']),
...                        table = 'NB_PRIOR_PROBABILITY_DEMO')

>>> # Change the setting parameter and refit the model
... # with a user-defined prior table.
... new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
... nb_mod = nb_mod.set_params(**new_setting).fit(train_x,
...                                               train_y,
...                                               priors = priors)

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_NAIVE_BAYES</td>
</tr>
<tr>
<td>CLAS_PRIORS_TABLE_NAME</td>
<td>&quot;OML_USER&quot;).&quot;NB_PRIOR_PROBABILITY_DEMO&quot;</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>NABS_PAIRWISE_THRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>NABS_SINGLETON_THRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_ROWS</td>
<td>104</td>
</tr>
</tbody>
</table>
Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

<table>
<thead>
<tr>
<th>TARGET_NAME</th>
<th>TARGET_VALUE</th>
<th>PRIOR_PROBABILITY</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Species</td>
<td>setosa</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>Species</td>
<td>versicolor</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>Species</td>
<td>virginica</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Conditionals:

<table>
<thead>
<tr>
<th>TARGET_NAME</th>
<th>TARGET_VALUE</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>CONDITIONAL_PROBABILITY</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>None</td>
<td>( ; 1.05]</td>
<td>1.05</td>
</tr>
<tr>
<td>1</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>None</td>
<td>(1.05; 1.2]</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>None</td>
<td>(1.2; 1.35]</td>
<td>1.35</td>
</tr>
<tr>
<td>3</td>
<td>Species</td>
<td>setosa</td>
<td>Petal_Length</td>
<td>None</td>
<td>(1.35; 1.45]</td>
<td>1.45</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>152</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>None</td>
<td>(3.25; 3.35]</td>
<td>3.25</td>
</tr>
<tr>
<td>153</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>None</td>
<td>(3.35; 3.45]</td>
<td>3.35</td>
</tr>
<tr>
<td>154</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>None</td>
<td>(3.55; 3.65]</td>
<td>3.55</td>
</tr>
<tr>
<td>155</td>
<td>Species</td>
<td>virginica</td>
<td>Sepal_Width</td>
<td>None</td>
<td>(3.75; 3.85]</td>
<td>3.75</td>
</tr>
</tbody>
</table>

[156 rows x 7 columns]

```python
>>> # Use the model to make predictions on test data.
>>> nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Species</th>
<th>PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>
>>> # Return the prediction probability.
>>> nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']]
...                proba = True)

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Species</th>
<th>PREDICTION</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>setosa</td>
<td>setosa</td>
<td>1.000000</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>setosa</td>
<td>setosa</td>
<td>1.000000</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>setosa</td>
<td>setosa</td>
<td>1.000000</td>
</tr>
<tr>
<td>5.8</td>
<td>4.0</td>
<td>setosa</td>
<td>setosa</td>
<td>1.000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6.7</td>
<td>3.3</td>
<td>virginica</td>
<td>virginica</td>
<td>1.000000</td>
</tr>
<tr>
<td>6.7</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>0.953848</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>1.000000</td>
</tr>
<tr>
<td>5.9</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>0.932334</td>
</tr>
</tbody>
</table>

>>> # Return the top two most influential attributes of the highest probability class.
>>> nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']]
...                topNAttrs = 2)

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Species</th>
<th>PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.7</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>virginica</td>
<td>virginica</td>
</tr>
</tbody>
</table>

TOP_N_ATTRIBUTES
0 <Details algorithm="Naive Bayes" class="setosa...>
1 <Details algorithm="Naive Bayes" class="setosa...>
2 <Details algorithm="Naive Bayes" class="setosa...>
3 <Details algorithm="Naive Bayes" class="setosa...>
...
42 <Details algorithm="Naive Bayes" class="virgin...>
43 <Details algorithm="Naive Bayes" class="virgin...>
44 <Details algorithm="Naive Bayes" class="virgin...>
45 <Details algorithm="Naive Bayes" class="virgin...>

>>> # Make predictions and return the probability for each class
>>> nb_mod.predict_proba(test_dat.drop('Species'),
...                      supplemental_cols = test_dat[:, ['Sepal_Length',
...                         'Species'])].sort_values(by =
...                            ['Sepal_Length',
...                             'Species',
...                             'PROBABILITY_OF_setosa,'
...                             'PROBABILITY_OF_versicolor'])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Species</th>
<th>PROBABILITY_OF_SETOSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1</td>
<td>4.4</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>4.8</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>6.7</td>
<td>virginica</td>
</tr>
<tr>
<td>43</td>
<td>6.9</td>
<td>versicolor</td>
</tr>
<tr>
<td>44</td>
<td>6.9</td>
<td>virginica</td>
</tr>
<tr>
<td>45</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROBABILITY_OF_VERSICOLOR</th>
<th>PROBABILITY_OF_VIRGINICA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.327306e-21</td>
</tr>
<tr>
<td>1</td>
<td>3.497737e-20</td>
</tr>
<tr>
<td>2</td>
<td>2.238553e-13</td>
</tr>
<tr>
<td>3</td>
<td>6.995487e-22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>4.741700e-13</td>
</tr>
<tr>
<td>43</td>
<td>1.778141e-07</td>
</tr>
<tr>
<td>44</td>
<td>2.963565e-20</td>
</tr>
<tr>
<td>45</td>
<td>4.156340e-01</td>
</tr>
</tbody>
</table>

>>> # Make predictions on new data and return the mean accuracy.
... nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.934783

**Neural Network**

The `oml.nn` class creates a Neural Network (NN) model for classification and regression. Neural Network models can be used to capture intricate nonlinear relationships between inputs and outputs or to find patterns in data.

The `oml.nn` class methods build a feed-forward neural network for regression on `oml.DataFrame` data. It supports multiple hidden layers with a specifiable number of nodes. Each layer can have one of several activation functions.

The output layer is a single numeric or binary categorical target. The output layer can have any of the activation functions. It has the linear activation function by default.

Modeling with the `ore.nn` class is well-suited for noisy and complex data such as sensor data. Problems that such data might have are the following:

- Potentially many (numeric) predictors, for example, pixel values
- The target may be discrete-valued, real-valued, or a vector of such values
- Training data may contain errors – robust to noise
- Fast scoring
- Model transparency is not required; models difficult to interpret
Typical steps in Neural Network modeling are the following:

1. Specifying the architecture
2. Preparing the data
3. Building the model
4. Specifying the stopping criteria: iterations, error on a validation set within tolerance
5. Viewing statistical results from the model
6. Improving the model

For information on the `oml.nn` class attributes and methods, invoke `help(oml.nn)` or `help(oml.hist)`, or see *Oracle Machine Learning for Python API Reference*.

**Settings for a Neural Network Model**

The following table lists settings for NN models.

**Table 5-12  Neural Network Models Settings**

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLAS_COST_TABLE_NAME         | `table_name`  | The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table.  
  • Column Name: ACTUAL_TARGET_VALUE  
    Data Type: Valid target data type
  • Column Name: PREDICTED_TARGET_VALUE  
    Data Type: Valid target data type
  • Column Name: COST  
    Data Type: NUMBER |
<p>| CLAS_WEIGHTS_BALANCED        | ON            | Indicates whether 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. |
|                             | OFF           |                                                                                                                                              |</p>
<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNET_ACTIVATIONS</td>
<td>A list of the following strings:</td>
<td>Defines the activation function for the hidden layers. For example, &quot;NNET_ACTIVATIONS_BIPOLAR_SIG&quot;, &quot;NNET_ACTIVATIONS_TANH&quot;. Different layers can have different activation functions. The number of activation functions must be consistent with NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER.</td>
</tr>
<tr>
<td></td>
<td>• &quot;NNET_ACTIVATIONS_ARCTAN&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;NNET_ACTIVATIONS_BIPOLAR_SIG&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;NNET_ACTIVATIONS_LINEAR&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;NNET_ACTIVATIONS_LOG_SIG&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;NNET_ACTIVATIONS_TANH&quot;</td>
<td></td>
</tr>
<tr>
<td>NNET_HELDASIDE_MAX_FAIL</td>
<td>A positive integer</td>
<td>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.</td>
</tr>
<tr>
<td>NNET_HELDASIDE_RATIO</td>
<td>0 &lt;= numeric_expr &lt;= 1</td>
<td>Defines the held ratio for the held-aside method. The default value is 0.25.</td>
</tr>
<tr>
<td>NNET_HIDDEN_LAYERS</td>
<td>A non-negative integer</td>
<td>Defines the topology by number of hidden layers. The default value is 1.</td>
</tr>
<tr>
<td>NNET_ITERATIONS</td>
<td>A positive integer</td>
<td>Specifies the maximum number of iterations in the Neural Network algorithm. The default value is 200.</td>
</tr>
<tr>
<td>NNET_NODES_PER_LAYER</td>
<td>A list of positive integers</td>
<td>Defines the topology by number of nodes per layer. Different layers can have different number of nodes. The value should be a comma separated list non-negative integers. For example, '10, 20, 5'. The setting values must be consistent with NNET_HIDDEN_LAYERS. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes &gt; 50).</td>
</tr>
<tr>
<td>NNET_REG_LAMBDA</td>
<td>TO_CHAR(numeric_expr &gt;= 0)</td>
<td>Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1.</td>
</tr>
</tbody>
</table>
Table 5-12  (Cont.) Neural Network Models Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNET_REGULARIZER</td>
<td>NNET_REGULARIZER_HELD ASIDE</td>
<td>Regularization setting for the Neural Network algorithm. If the total number of training rows is greater than 50000, then the default is NNET_REGULARIZER_HELD ASIDE. If the total number of training rows is less than or equal to 50000, then the default is NNET_REGULARIZER_NONE.</td>
</tr>
<tr>
<td></td>
<td>NNET_REGULARIZER_L2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NNET_REGULARIZER_NONE</td>
<td></td>
</tr>
<tr>
<td>NNET_SOLVER</td>
<td>NNET_SOLVER_ADAM</td>
<td>Specifies the method of optimization. The default value is NNET_SOLVER_LBFGS.</td>
</tr>
<tr>
<td></td>
<td>NNET_SOLVER_LBFGS</td>
<td></td>
</tr>
<tr>
<td>NNET_TOLERANCE</td>
<td>TO_CHAR(0 &lt; numeric_expr &lt; 1)</td>
<td>Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001.</td>
</tr>
<tr>
<td>NNET_WEIGHT_LOWER_BOUND</td>
<td>A real number</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• input layer dense attributes is ((1+\text{number of dense attributes}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• input layer sparse attributes is \text{number of sparse attributes}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• each hidden layer is ((1+\text{number of nodes in that hidden layer}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value of (r_nodes) is the number of nodes in the layer that the weight is connecting to.</td>
</tr>
<tr>
<td>NNET_WEIGHT_UPPER_BOUND</td>
<td>A real number</td>
<td>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.</td>
</tr>
<tr>
<td>ODMS_RANDOM_SEED</td>
<td>A non-negative integer</td>
<td>Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is 0.</td>
</tr>
</tbody>
</table>

See Also:

- About Model Settings
- Shared Settings
Example 5-15  Building a Neural Network Model

This example creates an NN model and uses some of the methods of the oml.nn class.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                            'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

dat = oml_iris.info().split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a Neural Network model object.
nn_mod = oml.nn(nnet_hidden_layers = 1,
                nnet_activations= 'NNET_ACTIVATIONS_LOG_SIG',
                NNET_NODES_PER_LAYER= '30')

# Fit the NN model according to the training data and parameter
# settings.
nn_mod = nn_mod.fit(train_x, train_y)

# Show details of the model.
nn_mod

# Use the model to make predictions on test data.
nn_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Petal_Length', 'Species']])

nn_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Species']], proba = True)

nn_mod.predict_proba(test_dat.drop('Species'),
                      supplemental_cols = test_dat[:, ['Sepal_Length',
                                                      'Sepal_Width',
                                                      'Petal_Length',
                                                      'Petal_Width',
                                                      'Species']])
```
Algorithm Name: Neural Network

Mining Function: CLASSIFICATION
Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME</td>
</tr>
<tr>
<td>1</td>
<td>CLAS_WEIGHTS_BALANCED</td>
</tr>
<tr>
<td>2</td>
<td>LBFGS_GRADIENT_TOLERANCE</td>
</tr>
<tr>
<td>3</td>
<td>LBFGS_HISTORY_DEPTH</td>
</tr>
<tr>
<td>4</td>
<td>LBFGS_SCALE_HESSIAN</td>
</tr>
<tr>
<td>5</td>
<td>NNET_ACTIVATIONS</td>
</tr>
<tr>
<td>6</td>
<td>NNET_HELDASIDE_MAX_FAIL</td>
</tr>
<tr>
<td>7</td>
<td>NNET_HELDASIDE_RATIO</td>
</tr>
<tr>
<td>8</td>
<td>NNET_HIDDEN_LAYERS</td>
</tr>
<tr>
<td>9</td>
<td>NNET_ITERATIONS</td>
</tr>
<tr>
<td>10</td>
<td>NNET_NODES_PER_LAYER</td>
</tr>
<tr>
<td>11</td>
<td>NNET_TOLERANCE</td>
</tr>
<tr>
<td>12</td>
<td>ODMS_DETAILS</td>
</tr>
<tr>
<td>13</td>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
</tr>
<tr>
<td>14</td>
<td>ODMS_RANDOM_SEED</td>
</tr>
<tr>
<td>15</td>
<td>ODMS_SAMPLING</td>
</tr>
<tr>
<td>16</td>
<td>PREP_AUTO</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NNET_REGULARIZER</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CONVERGED</td>
</tr>
<tr>
<td>1</td>
<td>ITERATIONS</td>
</tr>
<tr>
<td>2</td>
<td>LOSS_VALUE</td>
</tr>
<tr>
<td>3</td>
<td>NUM_ROWS</td>
</tr>
</tbody>
</table>

Attributes:

- Sepal_Length
- Sepal_Width
- Petal_Length
- Petal_Width

Partition: NO

Topology:

<table>
<thead>
<tr>
<th>HIDDEN_LAYER_ID</th>
<th>NUM_NODE</th>
<th>ACTIVATION_FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
</tbody>
</table>

Weights:

<table>
<thead>
<tr>
<th>LAYER</th>
<th>IDX_FROM</th>
<th>IDX_TO</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>Petal_Length</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>Petal_Length</td>
<td>None</td>
</tr>
</tbody>
</table>
None
2  0  0.0  2  Petal_Length
None            None
3  0  0.0  3  Petal_Length
None            None
...  ...  ...  ...            ...  ...
None            None
239  1  29.0  2  None
None            None
240  1  NaN  0  None
None            None
241  1  NaN  1  None
None            None
242  1  NaN  2  None
None            None

TARGET_VALUE   WEIGHT
0  None  -39.836487
1  None   32.604824
2  None   0.953903
3  None    0.714064
...  ...        ...
239  virginica  -22.650606
240  setosa    2.402457
241  versicolor  7.647615
242  virginica    -9.493982

[243 rows x 8 columns]

>>> # Use the model to make predictions on test data.
... nn_mod.predict(test_dat.drop(©Species©),
...     supplemental_cols = test_dat[:, [©Sepal_Length©, ©Sepal_Width©,
...                                      ©Petal_Length©, ©Species©]]

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Species</th>
<th>PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>44</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>virginica</td>
</tr>
<tr>
<td>45</td>
<td>6.7</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
</tr>
<tr>
<td>46</td>
<td>6.5</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
</tr>
<tr>
<td>47</td>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>virginica</td>
</tr>
</tbody>
</table>

>>> nn_mod.predict(test_dat.drop(©Species©),
...     supplemental_cols = test_dat[:, ['©Sepal_Length', '©Sepal_Width',
...                                      '©Petal_Length', '©Species']],
...     proba = True)

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Species</th>
<th>PREDICTION</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
<td>setosa</td>
<td>setosa</td>
<td>1.00000000</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>setosa</td>
<td>setosa</td>
<td>1.00000000</td>
</tr>
<tr>
<td>2</td>
<td>4.8</td>
<td>setosa</td>
<td>setosa</td>
<td>1.00000000</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>setosa</td>
<td>setosa</td>
<td>1.00000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
44  6.7  3.3 virginica virginica  1.000000
45  6.7  3.0 virginica virginica  1.000000
46  6.5  3.0 virginica virginica  1.000000
47  5.9  3.0 virginica virginica  1.000000

>>> nn_mod.predict_proba(test_dat.drop('Species'),
...                     supplemental_cols = test_dat[:, ['Sepal_Length', 'Species']]).sort_values(by = ['Sepal_Length', 'Species', 'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Species</th>
<th>PROBABILITY_OF_SETOSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.4</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>1</td>
<td>4.4</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>3</td>
<td>4.8</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>44</td>
<td>6.7</td>
<td>4.567318e-218</td>
</tr>
<tr>
<td>45</td>
<td>6.9</td>
<td>3.028266e-177</td>
</tr>
<tr>
<td>46</td>
<td>6.9</td>
<td>1.203417e-215</td>
</tr>
<tr>
<td>47</td>
<td>7.0</td>
<td>3.382837e-148</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROBABILITY_OF_VERSICOLOR</th>
<th>PROBABILITY_OF_VIRGINICA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.491272e-67</td>
</tr>
<tr>
<td></td>
<td>3.459448e-283</td>
</tr>
<tr>
<td>1</td>
<td>8.038930e-58</td>
</tr>
<tr>
<td></td>
<td>2.883999e-288</td>
</tr>
<tr>
<td>2</td>
<td>5.273544e-64</td>
</tr>
<tr>
<td></td>
<td>2.243282e-293</td>
</tr>
<tr>
<td>3</td>
<td>1.332150e-78</td>
</tr>
<tr>
<td></td>
<td>2.040723e-283</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>44</td>
<td>1.328042e-36</td>
</tr>
<tr>
<td></td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>45</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td></td>
<td>5.063405e-55</td>
</tr>
<tr>
<td>46</td>
<td>4.000953e-31</td>
</tr>
<tr>
<td></td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>47</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td></td>
<td>2.593761e-121</td>
</tr>
</tbody>
</table>

>>> nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.9375

>>> # Change the setting parameter and refit the model.
... new_setting = {'NNET_NODES_PER_LAYER': '50'}

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_NEURAL_NETWORK</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>LBFGS_GRADIENT_TOLERANCE</td>
<td>0.000000001</td>
</tr>
<tr>
<td>LBFGS_HISTORY_DEPTH</td>
<td>20</td>
</tr>
<tr>
<td>LBFGS_SCALE_HESSIAN</td>
<td>LBFGS_SCALE_HESSIAN_ENABLE</td>
</tr>
<tr>
<td>NNET_ACTIVATIONS</td>
<td>'NNET_ACTIVATIONS_LOG_SIG'</td>
</tr>
<tr>
<td>NNET_HELDASIDE_MAX_FAIL</td>
<td>6</td>
</tr>
<tr>
<td>NNET_HELDASIDE_RATIO</td>
<td>0.25</td>
</tr>
<tr>
<td>NNET_HIDDEN_LAYERS</td>
<td>1</td>
</tr>
</tbody>
</table>
Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNET_REGULARIZER</td>
<td>NNET_REGULARIZER_NONE</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVERGED</td>
<td>YES</td>
</tr>
<tr>
<td>ITERATIONS</td>
<td>68.0</td>
</tr>
<tr>
<td>LOSS_VALUE</td>
<td>0.0</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>102.0</td>
</tr>
</tbody>
</table>

Attributes:

Sepal_Length
Sepal_Width
Petal_Length
Petal_Width

Partition: NO

Topology:

<table>
<thead>
<tr>
<th>HIDDEN_LAYER_ID</th>
<th>NUM_NODE</th>
<th>ACTIVATION_FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

Weights:

<table>
<thead>
<tr>
<th>LAYER</th>
<th>IDX_FROM</th>
<th>IDX_TO</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0.0</td>
<td>Petal_Length</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>1</td>
<td>0.0</td>
<td>Petal_Length</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>2</td>
<td>0.0</td>
<td>Petal_Length</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>3</td>
<td>0.0</td>
<td>Petal_Length</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>399</td>
<td>1</td>
<td>49.0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>400</td>
<td>NaN</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>401</td>
<td>NaN</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>402</td>
<td>NaN</td>
<td>2</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The `oml.rf` class creates a Random Forest (RF) model that provides an ensemble learning technique for classification.

By combining the ideas of bagging and random selection of variables, the Random Forest algorithm produces a collection of decision trees with controlled variance while avoiding overfitting, which is a common problem for decision trees.

For information on the `oml.rf` class attributes and methods, invoke `help(oml.rf)` or see *Oracle Machine Learning for Python API Reference*.

### Settings for a Random Forest Model

The following table lists settings for RF models.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAS_COST_TABLE_NAME</td>
<td><code>table_name</code></td>
<td>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Column Name: COST Data Type: NUMBER</td>
</tr>
<tr>
<td>CLAS_MAX_SUP_BINS</td>
<td><code>2 &lt;= a number &lt;= 254</code></td>
<td>Specifies the maximum number of bins for each attribute. The default value is 32.</td>
</tr>
<tr>
<td>Setting Name</td>
<td>Setting Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>ON/OFF</td>
<td>Indicates whether 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.</td>
</tr>
<tr>
<td>ODMS_RANDOM_SEED</td>
<td>A non-negative integer</td>
<td>Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is 0.</td>
</tr>
<tr>
<td>RFOR_MTRY</td>
<td>A number &gt;= 0</td>
<td>Size of the random subset of columns to consider 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.</td>
</tr>
<tr>
<td>RFOR_NUM_TREES</td>
<td>1 &lt;= a number &lt;= 65535</td>
<td>Number of trees in the forest The default value is 20.</td>
</tr>
<tr>
<td>RFOR_SAMPLING_RATIO</td>
<td>0 &lt; a fraction &lt;= 1</td>
<td>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.</td>
</tr>
<tr>
<td>TREE_IMPURITY_METRIC</td>
<td>TREE_IMPURITY_ENTROPY/TREE_IMPURITY_GINI</td>
<td>Tree impurity metric for a decision tree model. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with 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.</td>
</tr>
<tr>
<td>TREE_TERM_MAXDEPTH</td>
<td>2 &lt;= a number &lt;= 100</td>
<td>Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). The default is 16.</td>
</tr>
</tbody>
</table>
Table 5-13  (Cont.) Random Forest Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>0 &lt;= a number &lt;= 10</td>
<td>The minimum number of training rows in a node expressed as a percentage of the rows in the training data. The default value is 0.05, indicating 0.05%.</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>0 &lt; a number &lt;= 20</td>
<td>Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>A number &gt;= 0</td>
<td>Minimum number of rows in a node. The default value is 10.</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>A number &gt; 1</td>
<td>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. The default value is 20.</td>
</tr>
</tbody>
</table>

See Also:
- About Model Settings
- Shared Settings

Example 5-16  Using the oml.rf Class

This example creates an RF model and uses some of the methods of the oml.rf class.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor', 2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop(table = 'RF_COST')
except:
    pass
```
# Create the IRIS database table and the proxy object for the table.
ml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0], ['setosa', 'virginica', 0.2], ['setosa', 'versicolor', 0.8], ['virginica', 'virginica', 0], ['virginica', 'setosa', 0.5], ['virginica', 'versicolor', 0.5], ['versicolor', 'versicolor', 0], ['versicolor', 'setosa', 0.4], ['versicolor', 'virginica', 0.6]]
cost_matrix = \
    oml.create(pd.DataFrame(cost_matrix,
                            columns = ['ACTUAL_TARGET_VALUE',
                                       'PREDICTED_TARGET_VALUE',
                                       'COST']),
                            table = 'RF_COST')

# Create an RF model object.
rf_mod = oml.rf(tree_term_max_depth = 2)

# Fit the RF model according to the training data and parameter settings.
rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Show details of the model.
rf_mod

# Use the model to make predictions on the test data.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                               'Sepal_Width',
                                               'Petal_Length',
                                               'Species']])

# Return the prediction probability.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                               'Sepal_Width',
                                               'Species']],
               proba = True)

# Return the top two most influential attributes of the highest probability class.
rf_mod.predict_proba(test_dat.drop('Species'),
supplemental_cols = test_dat[:, ['Sepal_Length', 'Species']],
        topN = 2).sort_values(by = ['Sepal_Length', 'Species'])

rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

# Reset TREE_TERM_MAX_DEPTH and refit the model.
rf_mod.set_params(tree_term_max_depth = 3).fit(train_x, train_y, cost_matrix)

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()

>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])

>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
...    oml.drop(table = 'RF_COST')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()

>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']

>>> test_dat = dat[1]

>>> # Create a cost matrix table in the database.
... cost_matrix = [
...                ['setosa', 'setosa', 0],
...                ['setosa', 'virginica', 0.2],
...                ['setosa', 'versicolor', 0.8],
...                ['virginica', 'virginica', 0],
...                ['virginica', 'setosa', 0.5],
...                ['virginica', 'versicolor', 0.5],
...                ['versicolor', 'versicolor', 0],
...                ['versicolor', 'setosa', 0.4],
...                ['versicolor', 'virginica', 0.6]]

>>> cost_matrix = \
...   oml.create(pd.DataFrame(cost_matrix,
...                           columns = ['ACTUAL_TARGET_VALUE'],
```
Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_RANDOM_FOREST</td>
</tr>
<tr>
<td>CLAS_COST_TABLE_NAME</td>
<td>&quot;OML_USER&quot;.&quot;RF_COST&quot;</td>
</tr>
<tr>
<td>CLAS_MAX_SUP_BINS</td>
<td>32</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_RANDOM_SEED</td>
<td>0</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
<tr>
<td>RFOR_NUM_TREES</td>
<td>20</td>
</tr>
<tr>
<td>RFOR_SAMPLING_RATIO</td>
<td>.5</td>
</tr>
<tr>
<td>TREE_IMPURITY_METRIC</td>
<td>TREE_IMPURITY_GINI</td>
</tr>
<tr>
<td>TREE_TERM_MAX_DEPTH</td>
<td>2</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>.05</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>.1</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>10</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>20</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFOR_MTRY</td>
<td>2</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG_DEPTH</td>
<td>2</td>
</tr>
<tr>
<td>AVG_NODECOUNT</td>
<td>3</td>
</tr>
<tr>
<td>MAX_DEPTH</td>
<td>2</td>
</tr>
<tr>
<td>MAX_NODECOUNT</td>
<td>2</td>
</tr>
<tr>
<td>MIN_DEPTH</td>
<td>2</td>
</tr>
<tr>
<td>MIN_NODECOUNT</td>
<td>2</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>104</td>
</tr>
</tbody>
</table>

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
<th>ATTRIBUTE_IMPORTANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Petal_Length</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Width</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Sepal_Length</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>Sepal_Width</td>
<td>None</td>
</tr>
</tbody>
</table>

>>> # Use the model to make predictions on the test data.
... rf_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Species</th>
<th>PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>virginica</td>
</tr>
<tr>
<td>43</td>
<td>6.7</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
</tr>
<tr>
<td>44</td>
<td>6.5</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
</tr>
<tr>
<td>45</td>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>virginica</td>
</tr>
</tbody>
</table>

>>> # Return the prediction probability.
... rf_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                proba = True)

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Species</th>
<th>PREDICTION</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.9</td>
<td>setosa</td>
<td>setosa</td>
<td>0.989130</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>setosa</td>
<td>setosa</td>
<td>0.989130</td>
</tr>
<tr>
<td>2</td>
<td>4.8</td>
<td>setosa</td>
<td>setosa</td>
<td>0.989130</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>setosa</td>
<td>setosa</td>
<td>0.950000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>6.7</td>
<td>virginica</td>
<td>virginica</td>
<td>0.501016</td>
</tr>
<tr>
<td>43</td>
<td>6.7</td>
<td>virginica</td>
<td>virginica</td>
<td>0.501016</td>
</tr>
<tr>
<td>44</td>
<td>6.5</td>
<td>virginica</td>
<td>virginica</td>
<td>0.501016</td>
</tr>
<tr>
<td>45</td>
<td>5.9</td>
<td>virginica</td>
<td>virginica</td>
<td>0.501016</td>
</tr>
</tbody>
</table>

>>> # Return the top two most influential attributes of the highest
... # probability class.
>>> rf_mod.predict_proba(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Species']],
...                topN = 2).sort_values(by = ['Sepal_Length', 'Species'])

<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Species</th>
<th>TOP_1</th>
<th>TOP_1_VAL</th>
<th>TOP_2</th>
<th>TOP_2_VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>setosa</td>
<td>1.4</td>
<td>0.989130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>setosa</td>
<td>1.5</td>
<td>0.989130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>setosa</td>
<td>1.6</td>
<td>0.989130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>setosa</td>
<td>1.2</td>
<td>0.950000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>virginica</td>
<td>5.7</td>
<td>0.501016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>virginica</td>
<td>5.2</td>
<td>0.501016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>virginica</td>
<td>5.2</td>
<td>0.501016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>virginica</td>
<td>5.1</td>
<td>0.501016</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGO_NAME</td>
<td>ALGO_RANDOM_FOREST</td>
</tr>
<tr>
<td>CLAS_COST_TABLE_NAME</td>
<td>&quot;OML_USER&quot;.&quot;RF_COST&quot;</td>
</tr>
<tr>
<td>CLAS_MAX_SUP_BINS</td>
<td>32</td>
</tr>
<tr>
<td>CLAS_WEIGHTS_BALANCED</td>
<td>OFF</td>
</tr>
<tr>
<td>ODMS_DETAILS</td>
<td>ODMS_ENABLE</td>
</tr>
<tr>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>ODMS_RANDOM_SEED</td>
<td>0</td>
</tr>
<tr>
<td>ODMS_SAMPLING</td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>PREP_AUTO</td>
<td>ON</td>
</tr>
<tr>
<td>RFOR_NUM_TREES</td>
<td>20</td>
</tr>
<tr>
<td>RFOR_SAMPLING_RATIO</td>
<td>.5</td>
</tr>
<tr>
<td>TREE_IMPURITY_METRIC</td>
<td>TREE_IMPURITY_GINI</td>
</tr>
<tr>
<td>TREE_TERM_MAX_DEPTH</td>
<td>3</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_NODE</td>
<td>.05</td>
</tr>
<tr>
<td>TREE_TERM_MINPCT_SPLIT</td>
<td>.1</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_NODE</td>
<td>10</td>
</tr>
<tr>
<td>TREE_TERM_MINREC_SPLIT</td>
<td>20</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFOR_MTRY</td>
<td>2</td>
</tr>
</tbody>
</table>
Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 AVG_DEPTH</td>
<td>3</td>
</tr>
<tr>
<td>1 AVG_NODECOUNT</td>
<td>5</td>
</tr>
<tr>
<td>2 MAX_DEPTH</td>
<td>3</td>
</tr>
<tr>
<td>3 MAX_NODECOUNT</td>
<td>6</td>
</tr>
<tr>
<td>4 MIN_DEPTH</td>
<td>3</td>
</tr>
<tr>
<td>5 MIN_NODECOUNT</td>
<td>4</td>
</tr>
<tr>
<td>6 NUM_ROWS</td>
<td>104</td>
</tr>
</tbody>
</table>

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
<th>ATTRIBUTE_IMPORTANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Petal_Length</td>
<td>None</td>
<td>0.501022</td>
</tr>
<tr>
<td>1 Petal_Width</td>
<td>None</td>
<td>0.568170</td>
</tr>
<tr>
<td>2 Sepal_Length</td>
<td>None</td>
<td>0.091617</td>
</tr>
<tr>
<td>3 Sepal_Width</td>
<td>None</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

Singular Value Decomposition

Use the oml.svd class to build a model for feature extraction.

The oml.svd class creates a model that uses the Singular Value Decomposition (SVD) algorithm for feature extraction. SVD performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices: U, V, and D. Columns of matrix V contain the right singular vectors and columns of matrix U contain the left singular vectors. Matrix D is a diagonal matrix and its singular values reflect the amount of data variance captured by the bases.

The SVDS_MAX_NUM_FEATURES constant specifies the maximum number of features supported by SVD. The value of the constant is 2500.

For information on the oml.svd class attributes and methods, invoke help(oml.svd) or see Oracle Machine Learning for Python API Reference.

Settings for a Singular Value Decomposition Model

Table 5-14  Singular Value Decomposition Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT_NUM_FEATURES</td>
<td>TO_CHAR(numeric_expr &gt;=1)</td>
<td>The number of features to extract.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is estimated by the algorithm. If the matrix rank is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>smaller than this number, fewer features are returned.</td>
</tr>
</tbody>
</table>
Table 5-14  (Cont.) Singular Value Decomposition Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVDS_OVER_SAMPLING</td>
<td>Range [1, 5000]</td>
<td>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. TSVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN.</td>
</tr>
<tr>
<td>SVDS_POWER_ITERATION</td>
<td>Range [0, 20]</td>
<td>Improves the accuracy of the SSVD solver. The default value is 2. SVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN.</td>
</tr>
<tr>
<td>SVDS_RANDOM_SEED</td>
<td>Range [0 - 4,294,967,296]</td>
<td>The random seed value for initializing the sampling matrix used by the Stochastic SVD solver. The default value is 0. SVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN.</td>
</tr>
<tr>
<td>SVDS_SCORING_MODE</td>
<td>SVDS_SCORING_PCA, SVDS_SCORING_SVD</td>
<td>Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections are the same as the U matrix. When the build data is scored with PCA, the projections are the product of the U and D matrices. The default value is SVDS_SCORING_SVD.</td>
</tr>
</tbody>
</table>
| SVDS_SOLVER               | SVDS_SOLVER_STEIGEN, SVDS_SOLVER_SSVD, SVDS_SOLVER_TSEIGEN, SVDS_SOLVER_TSSVD | Specifies the solver to be used for computing SVD of the data. For 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), which 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]   | Defines the minimum value for the eigenvalue of a feature as a share of the first eigenvalue to not prune. Use this setting to prune features. The default value is data driven. |
Table 5-14  (Cont.) Singular Value Decomposition Model Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVDS_U_MATRIX_OUTPUT</td>
<td></td>
<td>Specifies whether 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 <code>SVDS_U_MATRIX_OUTPUT</code> is enabled. When <code>SVDS_U_MATRIX_OUTPUT</code> 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. The default value is <code>SVDS_U_MATRIX_DISABLE</code>.</td>
</tr>
</tbody>
</table>

See Also:

- About Model Settings
- Shared Settings

Example 5-17  Using the oml.svd Class

This example uses some of the methods of the `oml.svd` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal Length', 'Sepal Width',
                            'Petal Length', 'Petal Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
                                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]
```
# Create an SVD model object.
svd_mod = oml.svd(ODMSDETAILS = 'ODMS_ENABLE')

# Fit the model according to the training data and parameter settings.
svd_mod = svd_mod.fit(train_dat)

# Show the model details.
svd_mod

# Use the model to make predictions on the test data.
svd_mod.predict(test_dat,
    supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])

# Perform dimensionality reduction and return values for the two features that have the highest topN values.
svd_mod.transform(test_dat,
    supplemental_cols = test_dat[:, ['Sepal_Length']],
    topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1', 'TOP_1_VAL'])

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...     columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...     {0: 'setosa', 1: 'versicolor', 2: 'virginica'}[x], iris.target)),
...     columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
```
>>> test_dat = dat[1]

# Create an SVD model object.
... svd_mod = oml.svd(ODMS_DETAILS = ODMS_ENABLE)

# Fit the model according to the training data and parameter settings.
... svd_mod = svd_mod.fit(train_dat)

# Show the model details.
... svd_mod

Algorithm Name: Singular Value Decomposition

Mining Function: FEATURE_EXTRACTION

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME</td>
</tr>
<tr>
<td>1</td>
<td>ALGO_SINGULAR_VALUE_DECOMP</td>
</tr>
<tr>
<td>2</td>
<td>ODMS_DETAILS</td>
</tr>
<tr>
<td>3</td>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
</tr>
<tr>
<td>4</td>
<td>ODMS_SAMPLING</td>
</tr>
<tr>
<td>5</td>
<td>PREP_AUTO</td>
</tr>
<tr>
<td>6</td>
<td>SVDS_SCORING_MODE</td>
</tr>
<tr>
<td>7</td>
<td>SVDS_U_MATRIX_OUTPUT</td>
</tr>
</tbody>
</table>

Computed Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FEAT_NUM_FEATURES</td>
</tr>
<tr>
<td>1</td>
<td>SVDS_SOLVER</td>
</tr>
<tr>
<td>2</td>
<td>SVDS_TOLERANCE</td>
</tr>
</tbody>
</table>

Global Statistics:

<table>
<thead>
<tr>
<th>attribute name</th>
<th>attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_COMPONENTS</td>
<td>8</td>
</tr>
<tr>
<td>NUM_ROWS</td>
<td>111</td>
</tr>
<tr>
<td>SUGGESTED_CUTOFF</td>
<td>1</td>
</tr>
</tbody>
</table>

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Features:

<table>
<thead>
<tr>
<th>FEATURE_ID</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ID</td>
<td>None</td>
<td>0.996297</td>
</tr>
<tr>
<td>1</td>
<td>Petal_Length</td>
<td>None</td>
<td>0.046646</td>
</tr>
<tr>
<td>2</td>
<td>Petal_Width</td>
<td>None</td>
<td>0.015917</td>
</tr>
<tr>
<td>3</td>
<td>Sepal_Length</td>
<td>None</td>
<td>0.063312</td>
</tr>
</tbody>
</table>

... ... ... ... ...
>>> # Use the model to make predictions on the test data.
>>> svd_mod.predict(test_dat,
...                supplemental_cols = test_dat[:,
...                                                'Sepal_Length',
...                                                'Sepal_Width',
...                                                'Petal_Length',
...                                                'Species'])
Sepal_Length  Sepal_Width  Petal_Length     Species  FEATURE_ID
0            5.0          3.6           1.4    setosa           2
1            5.0          3.4           1.5    setosa           2
2            4.4          2.9           1.4    setosa           8
3            4.9          3.1           1.5    setosa           2
... ... ... ... ... ...
35           6.9          3.1           5.4 virginica           1

Chapter 5
Singular Value Decomposition
Support Vector Machine

The `oml.svm` class creates a Support Vector Machine (SVM) model for classification, regression, or anomaly detection.

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.

SVM models have a functional form similar to neural networks and radial basis functions, which are both popular machine learning techniques.

SVM can be used to solve the following problems:

- **Classification**: SVM classification is based on decision planes that define decision boundaries. A decision plane is one that separates a set of objects having different class memberships. SVM finds the vectors (“support vectors”) that define the separators that give the widest separation of classes.

SVM classification supports both binary and multiclass targets.

- **Regression**: 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.

- **Anomaly Detection**: Anomaly detection identifies unusual cases in data that is seemingly homogeneous. Anomaly detection is an important tool for detecting fraud, network intrusion, and other rare events that may have great significance but are hard to find.

Anomaly detection is implemented as one-class SVM classification. An anomaly detection model predicts whether a data point is typical for a given distribution or not.
The `oml.svm` class builds each of these three different types of models. Some arguments apply to classification models only, some to regression models only, and some to anomaly detection models only.

For information on the `oml.svm` class attributes and methods, invoke `help(oml.svm)` or see *Oracle Machine Learning for Python API Reference*.

### Support Vector Machine Model Settings

The following table lists settings for SVM models.

Table 5-15  Support Vector Machine Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLAS_COST_TABLE_NAME  | `table_name`         | The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications.  
The cost matrix table is user-created. The following are the column requirements for the table.  
- Column Name: `ACTUAL_TARGET_VALUE`  
  Data Type: Valid target data type  
- Column Name: `PREDICTED_TARGET_VALUE`  
  Data Type: Valid target data type  
- Column Name: `COST`  
  Data Type: `NUMBER` |
| CLAS_WEIGHTS_BALANCED | `ON` `OFF`           | Indicates whether 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`. |
| CLAS_WEIGHTS_TABLE_NAME | `table_name`       | The name of a table that stores weighting information for individual target values in 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. The following are the column requirements for the table.  
- Column Name: `TARGET_VALUE`  
  Data Type: Valid target data type  
- Column Name: `CLASS_WEIGHT`  
  Data Type: `NUMBER` |
| SVMS_BATCH_ROWS       | Positive integer     | Sets the size of the batch for the SGD solver. This setting applies to SVM models with linear kernel. An input of 0 triggers a data driven batch size estimate. The default value is 20000. |
| 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. |
Table 5-15  (Cont.) Support Vector Machine Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMS_EPSILON</td>
<td>TO_CHAR(numeric_expr &gt;0)</td>
<td>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.</td>
</tr>
<tr>
<td>SVMS_KERNEL_FUNCTION</td>
<td>SVMS_GAUSSIAN</td>
<td>Kernel for Support Vector Machine. Linear or Gaussian.</td>
</tr>
<tr>
<td></td>
<td>SVMS_LINEAR</td>
<td>The default value is SVMS_LINEAR.</td>
</tr>
<tr>
<td>SVMS_NUM_ITERATIONS</td>
<td>Positive integer</td>
<td>Sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.</td>
</tr>
<tr>
<td>SVMS_NUM_PIVOTS</td>
<td>Range [1; 10000]</td>
<td>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.</td>
</tr>
<tr>
<td>SVMS_OUTLIER_RATE</td>
<td>TO_CHAR(0&lt;numeric_expr &lt;1)</td>
<td>The desired rate of outliers in the training data. Valid for One-Class SVM models only (Anomaly Detection). The default value is 0.01.</td>
</tr>
<tr>
<td>SVMS_REGULARIZER</td>
<td>SVMS_REGULARIZER_L1</td>
<td>Controls the type of regularization that the SGD SVM solver uses. The setting applies only to linear SVM models. The default value is system determined because it depends on the potential model size.</td>
</tr>
<tr>
<td></td>
<td>SVMS_REGULARIZER_L2</td>
<td></td>
</tr>
<tr>
<td>SVMS_SOLVER</td>
<td>SVMS_SOLVER_SGD (Sub-Gradient Descend)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>SVMS_SOLVER_IPM (Interior Point Method)</td>
<td></td>
</tr>
<tr>
<td>SVMS_STD_DEV</td>
<td>TO_CHAR(numeric_expr &gt;0)</td>
<td>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 the Gaussian kernel. The default value is estimated from the data by the algorithm.</td>
</tr>
</tbody>
</table>

See Also:

- About Model Settings
- Shared Settings
Example 5-18  Using the oml.svm Class

This example demonstrates the use of various methods of the oml.svm class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```python
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                            'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                            {0: 'setosa', 1: 'versicolor',
                             2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create an SVM model object.
svm_mod = oml.svm('classification',
                   svms_kernel_function =
                   'dbms_data_mining.svms_linear')

# Fit the SVM Model according to the training data and parameter
# settings.
svm_mod.fit(train_x, train_y)

# Use the model to make predictions on test data.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                        'Sepal_Width',
                                        'Petal_Length',
                                        'Species']])

# Return the prediction probability.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                        'Sepal_Width',
                                        'Species']],
                proba = True)
```
svm_mod.predict_proba(test_dat.drop('Species'),
    supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width', 'Species']],
    topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])

svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> try:
...    oml.drop('IRIS')
... except:
...    pass

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

>>> # Create an SVM model object.
... svm_mod = oml.svm('classification',
...                   svms_kernel_function =
...                   'dbms_data_mining.svms_linear')

>>> # Fit the SVM model according to the training data and parameter
... # settings.
>>> svm_mod.fit(train_x, train_y)

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: Species

Settings:
setting name                      setting value
0                     ALGO_NAME  ALGO_SUPPORT VECTOR MACHINES
1         CLAS_WEIGHTS_BALANCED                           OFF
2                  ODMS_DETAILS                   ODMS_ENABLE
3  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
4                ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
5                     PREP_AUTO                            ON
6        SVMS_CONV_TOLERANCE                         .0001
7     SVMS KERNEL FUNCTION                     SVMS_LINEAR

Computed Settings:
setting name    setting value
0  SVMS COMPLEXITY_FACTOR               10
1     SVMS_NUM_ITERATIONS               30
2              SVMS_SOLVER  SVMS_SOLVER_IPM

Global Statistics:
attribute name  attribute value
0       CONVERGED              YES
1      ITERATIONS               14
2        NUM_ROWS              104

Attributes:
Petal Length
Petal Width
Sepal Length
Sepal Width

Partition: NO

COEFFICIENTS:

<table>
<thead>
<tr>
<th>TARGET_VALUE</th>
<th>ATTRIBUTE_NAME</th>
<th>ATTRIBUTE_SUBNAME</th>
<th>ATTRIBUTE_VALUE</th>
<th>COEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>Petal Length</td>
<td>None</td>
<td>None</td>
<td>-0.5809</td>
</tr>
<tr>
<td>setosa</td>
<td>Petal Width</td>
<td>None</td>
<td>None</td>
<td>-0.7736</td>
</tr>
<tr>
<td>setosa</td>
<td>Sepal Length</td>
<td>None</td>
<td>None</td>
<td>-0.1653</td>
</tr>
<tr>
<td>setosa</td>
<td>Sepal Width</td>
<td>None</td>
<td>None</td>
<td>0.5689</td>
</tr>
<tr>
<td>setosa</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>-0.7355</td>
</tr>
<tr>
<td>versicolor</td>
<td>Petal Length</td>
<td>None</td>
<td>None</td>
<td>1.1304</td>
</tr>
<tr>
<td>versicolor</td>
<td>Petal Width</td>
<td>None</td>
<td>None</td>
<td>-0.3323</td>
</tr>
<tr>
<td>versicolor</td>
<td>Sepal Length</td>
<td>None</td>
<td>None</td>
<td>-0.8877</td>
</tr>
<tr>
<td>versicolor</td>
<td>Sepal Width</td>
<td>None</td>
<td>None</td>
<td>-1.2582</td>
</tr>
<tr>
<td>versicolor</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>-0.9091</td>
</tr>
<tr>
<td>virginica</td>
<td>Petal Length</td>
<td>None</td>
<td>None</td>
<td>4.6042</td>
</tr>
<tr>
<td>virginica</td>
<td>Petal Width</td>
<td>None</td>
<td>None</td>
<td>4.0681</td>
</tr>
<tr>
<td>virginica</td>
<td>Sepal Length</td>
<td>None</td>
<td>None</td>
<td>-0.7985</td>
</tr>
<tr>
<td>virginica</td>
<td>Sepal Width</td>
<td>None</td>
<td>None</td>
<td>-0.4328</td>
</tr>
<tr>
<td>virginica</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>-5.3180</td>
</tr>
</tbody>
</table>

>>> # Use the model to make predictions on test data.
... svm_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal Length',
...                                                 'Sepal Width',
...                                                 'Petal Length',
...                                                 'Species']])
<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Species</th>
<th>PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.7</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>5.2</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>virginica</td>
<td>virginica</td>
</tr>
</tbody>
</table>

```python
>>> # Return the prediction probability.
... svm_mod.predict(test_dat.drop('Species'),
... supplemental_cols = test_dat[:, ['Sepal_Length',
... 'Sepal_Width',
... 'Species']],
... proba = True)
<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Species</th>
<th>PREDICTION</th>
<th>PROBABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>setosa</td>
<td>setosa</td>
<td>0.761886</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>setosa</td>
<td>setosa</td>
<td>0.805510</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>setosa</td>
<td>setosa</td>
<td>0.920317</td>
</tr>
<tr>
<td>5.8</td>
<td>4.0</td>
<td>setosa</td>
<td>setosa</td>
<td>0.998398</td>
</tr>
<tr>
<td>6.7</td>
<td>3.3</td>
<td>virginica</td>
<td>virginica</td>
<td>0.927706</td>
</tr>
<tr>
<td>6.7</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>0.855353</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>0.799556</td>
</tr>
<tr>
<td>5.9</td>
<td>3.0</td>
<td>virginica</td>
<td>virginica</td>
<td>0.688024</td>
</tr>
</tbody>
</table>
```

```python
>>> # Make predictions and return the probability for each class
... # on new data.
... svm_mod.predict_proba(test_dat.drop('Species'),
... supplemental_cols = test_dat[:, ['Sepal_Length',
... 'Sepal_Width',
... 'Species']],
... topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])
<table>
<thead>
<tr>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Species</th>
<th>TOP_1</th>
<th>TOP_1_VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>3.0</td>
<td>setosa</td>
<td>setosa</td>
<td>0.690967</td>
</tr>
<tr>
<td>4.4</td>
<td>3.2</td>
<td>setosa</td>
<td>setosa</td>
<td>0.815643</td>
</tr>
<tr>
<td>4.3</td>
<td>3.2</td>
<td>setosa</td>
<td>setosa</td>
<td>0.605105</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>setosa</td>
<td>setosa</td>
<td>0.920317</td>
</tr>
<tr>
<td>6.7</td>
<td>3.3</td>
<td>virginica</td>
<td>virginica</td>
<td>0.927706</td>
</tr>
<tr>
<td>6.9</td>
<td>3.1</td>
<td>versicolor</td>
<td>versicolor</td>
<td>0.378391</td>
</tr>
<tr>
<td>6.9</td>
<td>3.1</td>
<td>virginica</td>
<td>virginica</td>
<td>0.881118</td>
</tr>
<tr>
<td>7.0</td>
<td>3.2</td>
<td>versicolor</td>
<td>setosa</td>
<td>0.586693</td>
</tr>
</tbody>
</table>
```

```python
>>> svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.895833
```
Automated Machine Learning

Use the automated algorithm selection, feature selection, and hyperparameter tuning of Automated Machine Learning to accelerate the machine learning modeling process.

Automated Machine Learning in OML4Py is described in the following topics:

- About Automated Machine Learning
- Algorithm Selection
- Feature Selection
- Model Tuning
- Model Selection

About Automated Machine Learning

Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.

Any modeling problem for a specified data set and prediction task involves a sequence of data cleansing and preprocessing, algorithm selection, and model tuning tasks. Each of these steps require data science expertise to help guide the process to an efficient final model. Automated Machine Learning (AutoML) automates this process with its built-in data science expertise.

OML4Py has the following AutoML capabilities:

- Automated algorithm selection that selects the appropriate algorithm from the supported machine learning algorithms
- Automated feature selection that reduces the size of the original feature set to speed up model training and tuning, while possibly also increasing model quality
- Automated tuning of model hyperparameters, which selects the model with the highest score metric from among several metrics as selected by the user

AutoML performs those common modeling tasks automatically, with less effort and potentially better results. It also leverages in-database algorithm parallel processing and scalability to minimize runtime and produce high-quality results.

**Note:**

As the `fit` method of the machine learning classes does, the AutoML functions `reduce`, `select`, and `tune` provide a `case_id` parameter that you can use to achieve repeatable data sampling and data shuffling during model building.

The AutoML functionality is also available in a no-code user interface alongside OML Notebooks on Oracle Autonomous Database. For more information, see Oracle Machine Learning AutoML User Interface.
Automated Machine Learning Classes and Algorithms

The Automated Machine Learning classes are the following:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.automl.AlgorithmSelection</td>
<td>Using only the characteristics of the data set and the task, automatically selects the best algorithms from the set of supported Oracle Machine Learning algorithms. Supports classification and regression functions.</td>
</tr>
<tr>
<td>oml.automl.FeatureSelection</td>
<td>Uses meta-learning to quickly identify the most relevant feature subsets given a training data set and an Oracle Machine Learning algorithm. Supports classification and regression functions.</td>
</tr>
<tr>
<td>oml.automl.ModelTuning</td>
<td>Uses a highly parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the algorithm hyperparameters. Supports classification and regression functions.</td>
</tr>
<tr>
<td>oml.automl.ModelSelection</td>
<td>Selects the best Oracle Machine Learning algorithm and then tunes that algorithm. Supports classification and regression functions.</td>
</tr>
</tbody>
</table>

The Oracle Machine Learning algorithms supported by AutoML are the following:

<table>
<thead>
<tr>
<th>Algorithm Abbreviation</th>
<th>Algorithm Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>Decision Tree</td>
</tr>
<tr>
<td>glm</td>
<td>Generalized Linear Model</td>
</tr>
<tr>
<td>glm_ridge</td>
<td>Generalized Linear Model with ridge regression</td>
</tr>
<tr>
<td>nb</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>nn</td>
<td>Neural Network</td>
</tr>
<tr>
<td>rf</td>
<td>Random Forest</td>
</tr>
<tr>
<td>svm_gaussian</td>
<td>Support Vector Machine with Gaussian kernel</td>
</tr>
<tr>
<td>svm_linear</td>
<td>Support Vector Machine with linear kernel</td>
</tr>
</tbody>
</table>

Classification and Regression Metrics

The following tables list the scoring metrics supported by AutoML.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>Calculates the rate of correct classification of the target.</td>
</tr>
<tr>
<td></td>
<td>sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)</td>
</tr>
<tr>
<td></td>
<td><strong>Formula</strong>: $\frac{(tp + tn)}{samples}$</td>
</tr>
</tbody>
</table>

Table 6-1 Machine Learning Algorithms Supported by AutoML

Table 6-2 Binary and Multiclass Classification Metrics
### Table 6-2  (Cont.) Binary and Multiclass Classification Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1_macro</td>
<td>Calculates the f-score or f-measure, which is a weighted average of the precision and recall. The f1_macro takes the unweighted average of per-class scores.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$</td>
</tr>
<tr>
<td>f1_micro</td>
<td>Calculates the f-score or f-measure with micro-averaging in which true positives, false positives, and false negatives are counted globally.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$</td>
</tr>
<tr>
<td>f1_weighted</td>
<td>Calculates the f-score or f-measure with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$</td>
</tr>
<tr>
<td>precision_macro</td>
<td>Calculates the ability of the classifier to not label a sample incorrectly. The precision_macro takes the unweighted average of per-class scores.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $\frac{\text{tp}}{\text{tp + fp}}$</td>
</tr>
<tr>
<td>precision_micro</td>
<td>Calculates the ability of the classifier to not label a sample incorrectly. Uses micro-averaging in which true positives, false positives, and false negatives are counted globally.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $\frac{\text{tp}}{\text{tp + fp}}$</td>
</tr>
<tr>
<td>precision_weighted</td>
<td>Calculates the ability of the classifier to not label a sample incorrectly. Uses weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.</td>
</tr>
<tr>
<td></td>
<td><code>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Formula:</strong> $\frac{\text{tp}}{\text{tp + fp}}$</td>
</tr>
</tbody>
</table>
Table 6-2  (Cont.) Binary and Multiclass Classification Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
</table>
| recall_macro | Calculates the ability of the classifier to correctly label each class. The recall_macro takes the unweighted average of per-class scores.  
sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)  
Formula: tp / (tp + fn) | |
| recall_micro | Calculates the ability of the classifier to correctly label each class with micro-averaging in which the true positives, false positives, and false negatives are counted globally.  
sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)  
Formula: tp / (tp + fn) | |
| recall_weighted | Calculates the ability of the classifier to correctly label each class with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.  
sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)  
Formula: tp / (tp + fn) | |

See Also: [Scikit-learn classification metrics](#)

Table 6-3  Binary Classification Metrics Only

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
</table>
| f1           | Calculates the f-score or f-measure, which is a weighted average of the precision and recall. This metric by default requires a positive target to be encoded as 1 to function as expected.  
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)  
Formula: 2 * (precision * recall) / (precision + recall) | |
| precision    | Calculates the ability of the classifier to not label a sample positive (1) that is actually negative (0).  
sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)  
Formula: tp / (tp + fp) | |
Table 6-3  (Cont.) Binary Classification Metrics Only

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>recall</td>
<td>Calculates the ability of the classifier to label all positive (1) samples correctly.</td>
</tr>
<tr>
<td></td>
<td>sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)</td>
</tr>
<tr>
<td></td>
<td>Formula: $\frac{tp}{tp + fn}$</td>
</tr>
<tr>
<td>roc_auc</td>
<td>Calculates the Area Under the Receiver Operating Characteristic Curve (roc_auc) from prediction scores.</td>
</tr>
<tr>
<td></td>
<td>sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)</td>
</tr>
<tr>
<td></td>
<td>See also the definition of receiver operation characteristic.</td>
</tr>
</tbody>
</table>

Table 6-4  Regression Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>Calculates the coefficient of determination (R squared).</td>
</tr>
<tr>
<td></td>
<td>sklearn.metrics.r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</td>
</tr>
<tr>
<td></td>
<td>See also the definition of coefficient of determination.</td>
</tr>
<tr>
<td>neg_mean_absolute_error</td>
<td>Calculates the mean of the absolute difference of predicted and true targets (MAE).</td>
</tr>
<tr>
<td></td>
<td>sklearn.metrics.mean_absolute_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</td>
</tr>
<tr>
<td></td>
<td>Formula: $-\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y}_i)$</td>
</tr>
<tr>
<td>neg_mean_squared_error</td>
<td>Calculates the mean of the squared difference of predicted and true targets.</td>
</tr>
<tr>
<td></td>
<td>$-1.0 \times$ sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</td>
</tr>
<tr>
<td></td>
<td>Formula: $-\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2$</td>
</tr>
</tbody>
</table>
Table 6-4  (Cont.) Regression Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description, Scikit-learn Equivalent, and Formula</th>
</tr>
</thead>
</table>
| neg_mean_squared_log_error       | Calculates the mean of the difference in the natural log of predicted and true targets.  
|                                  | sklearn.metrics.mean_squared_log_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')  
|                                  | Formula: \[- \frac{1}{n} \sum_{i=1}^{n} (\log (Y_i) - \log (\hat{Y}_i))^2\] |
| neg_median_absolute_error        | Calculates the median of the absolute difference between predicted and true targets.  
|                                  | sklearn.metrics.median_absolute_error(y_true, y_pred)  
|                                  | Formula: \[-Med([Y_i - \hat{Y}_i, 0 \leq i < n])\] |

See Also: Scikit-learn regression metrics

Algorithm Selection

The oml.automl.AlgorithmSelection class uses the characteristics of the data set and the task to rank algorithms from the set of supported Oracle Machine Learning algorithms.

Selecting the best Oracle Machine Learning algorithm for a data set and a prediction task is non-trivial. No single algorithm works best for all modeling problems. The oml.automl.AlgorithmSelection class ranks the candidate algorithms according to how likely each is to produce a quality model. This is achieved by using Oracle advanced meta-learning intelligence learned from a repertoire of data sets with the goal of avoiding exhaustive searches, thereby reducing overall compute time and costs.

The oml.automl.AlgorithmSelection class supports classification and regression algorithms. To use the class, you specify a data set and the number of algorithms you want to evaluate.

The select method of the class returns a sorted list of the top algorithms and their predicted rank (from best to worst).

For information on the parameters and methods of the class, invoke help(oml.automl.AlgorithmSelection) or see Oracle Machine Learning for Python API Reference.

Example 6-1  Using the oml.automl.AlgorithmSelection Class

This example creates an oml.automl.AlgorithmSelection object and then displays the algorithm rankings with their corresponding score metric. You may select the top
entry or choose a different model depending on the needs of your particular business problem.

```python
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
b = datasets.load_breast_cancer()
b_data = b.data.astype(float)
X = pd.DataFrame(b_data, columns = b.feature_names)
y = pd.DataFrame(b.target, columns = ['TARGET'])

# Create the database table BreastCancer.
onl_df = oml.create(pd.concat([X, y], axis=1),
                    table = 'BreastCancer')

# Split the data into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated algorithm selection object with f1_macro as
# the score_metric argument.
asel = automl.AlgorithmSelection(mining_function='classification',
                                  score_metric='f1_macro', parallel=4)

# Run algorithm selection to get the top k predicted algorithms and
# their ranking without tuning.
algo_ranking = asel.select(X, y, k=3)

# Show the selected and tuned model.
[(m, '{:.2f}'.format(s)) for m, s in algo_ranking]

# Drop the database table.
onl.drop('BreastCancer')
```

**Listing for This Example**

```python
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])

>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                      table = 'BreastCancer')
```
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> # Create an automated algorithm selection object with f1_macro as
... # the score_metric argument.
... asel = automl.AlgorithmSelection(mining_function='classification',
... score_metric='f1_macro', parallel=4)

>>> # Run algorithm selection to get the top k predicted algorithms
... and
... their ranking without tuning.
... algo_ranking = asel.select(X, y, k=3)

>>> # Show the selected and tuned model.
>>> [(m, '{:.2f}'.format(s)) for m,s in algo_ranking]
[('svm_gaussian', '0.97'), ('glm_ridge', '0.96'), ('nn', '0.96')]

>>> # Drop the database table.
... oml.drop('BreastCancer')

Feature Selection

The `oml.automl.FeatureSelection` class identifies the most relevant feature subsets for a training data set and an Oracle Machine Learning algorithm.

In a data analytics application, feature selection is a critical data preprocessing step that has a high impact on both runtime and model performance. The `oml.automl.FeatureSelection` class automatically selects the most relevant features for a data set and model. It internally uses several feature-ranking algorithms to identify the best feature subset that reduces model training time without compromising model performance. Oracle advanced meta-learning techniques quickly prune the search space of this feature selection optimization.

The `oml.automl.FeatureSelection` class supports classification and regression algorithms. To use the `oml.automl.FeatureSelection` class, you specify a data set and the Oracle Machine Learning algorithm on which to perform the feature reduction.

For information on the parameters and methods of the class, invoke `help(oml.automl.FeatureSelection)` or see `Oracle Machine Learning for Python API Reference`.

Example 6-2 Using the `oml.automl.FeatureSelection` Class

This example uses the `oml.automl.FeatureSelection` class. The example builds a model on the full data set and computes predictive accuracy. It performs automated feature selection, filters the columns according to the determined set, and rebuilds the model. It then recomputes predictive accuracy.

```python
import oml
from oml import automl
import pandas as pd
import numpy as np
```
from sklearn import datasets

# Load the digits data set into the database.
digits = datasets.load_digits()
X = pd.DataFrame(digits.data,
    columns = ['pixel'{}.format(i) for i
        in range(digits.data.shape[1])])
y = pd.DataFrame(digits.target, columns = ['digit'])
oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')

# Split the data set into train and test.
train, test = oml_df.split(ratio=(0.8, 0.2),
    seed = 1234, strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
X_test, y_test = test.drop('digit'), test['digit']

# Default model performance before feature selection.
mod = oml.svm(mining_function='classification').fit(X_train, y_train)
"{:.2}".format(mod.score(X_test, y_test))

# Create an automated feature selection object with accuracy
# as the score_metric.
fs = automl.FeatureSelection(mining_function='classification',
    score_metric='accuracy', parallel=4)

# Get the reduced feature subset on the train data set.
subset = fs.reduce('svm_linear', X_train, y_train)
"{} features reduced to {}".format(len(X_train.columns),
    len(subset))

# Use the subset to select the features and create a model on the
# new reduced data set.
X_new = X_train[:,subset]
X_test_new = X_test[:,subset]
mod = oml.svm(mining_function='classification').fit(X_new, y_train)
"{:.2} with {:.1f}x feature reduction".format(
    mod.score(X_test_new, y_test),
    len(X_train.columns)/len(X_new.columns))

# For reproducible results, add a case_id column with unique row
# identifiers.
row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
    columns = ['CASE_ID'])
oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
    table = 'DIGITS_CID')

train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
    hash_cols='CASE_ID',
    strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
X_test, y_test = test.drop('digit'), test['digit']
# Provide the case_id column name to the feature selection
# reduce function.
subset = fs.reduce('svm_linear', X_train,
                   y_train, case_id='CASE_ID')
"{} features reduced to {} with case_id".format(
    len(X_train.columns)-1,
    len(subset))

# Drop the tables created in the example.
oml.drop('DIGITS')
oml.drop('DIGITS_CID')

Listing for This Example

```python
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets

>>> # Load the digits data set into the database.
... digits = datasets.load_digits()
>>> X = pd.DataFrame(digits.data,
...                   columns = ['pixel{}'.format(i) for i
...                         in range(digits.data.shape[1])])
>>> y = pd.DataFrame(digits.target, columns = ['digit'])
>>> oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')

>>> # Split the data set into train and test.
... train, test = oml_df.split(ratio=(0.8, 0.2),
...                            seed = 1234, strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']

>>> # Default model performance before feature selection.
... mod = oml.svm(mining_function='classification').fit(X_train,
...                                                     y_train)
... "{:.2}".format(mod.score(X_test, y_test))
'0.92'

>>> # Create an automated feature selection object with accuracy
... # as the score_metric.
... fs = automl.FeatureSelection(mining_function='classification',
...                               score_metric='accuracy', parallel=4)

>>> # Get the reduced feature subset on the train data set.
... subset = fs.reduce('svm_linear', X_train, y_train)
... "{} features reduced to {}".format(len(X_train.columns),
...                                      len(subset))
'64 features reduced to 41'

>>> # Use the subset to select the features and create a model on the
... # new reduced data set.
... X_new = X_train[:,subset]
```
>>> X_test_new = X_test[:, subset]
>>> mod = oml.svm(mining_function='classification').fit(X_new, y_train)
>>> '{:.2} with {:.1f}x feature reduction'.format(
...     mod.score(X_test_new, y_test),
...     len(X_train.columns)/len(X_new.columns))
'0.92 with 1.6x feature reduction'

>>> # Drop the DIGITS table.
... oml.drop('DIGITS')

>>> # For reproducible results, add a case_id column with unique row
... # identifiers.
>>> row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
...     columns = ['CASE_ID'])
>>> oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
...     table = 'DIGITS_CID')

>>> train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
...     hash_cols='CASE_ID',
...     strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']

>>> # Provide the case_id column name to the feature selection
... # reduce function.
>>> subset = fs.reduce('svm_linear', X_train,
...     y_train, case_id='CASE_ID')
>>> '{} features reduced to {} with case_id'.format(
...     len(X_train.columns)-1,
...     len(subset))
'64 features reduced to 45 with case_id'

>>> # Drop the tables created in the example.
... oml.drop('DIGITS')
>>> oml.drop('DIGITS_CID')

Related Topics

- About Automated Machine Learning
  Automated Machine Learning (AutoML) provides built-in data science expertise about
data analytics and modeling that you can employ to build machine learning models.

Model Tuning

The `oml.automl.ModelTuning` class tunes the hyperparameters for the specified
classification or regression algorithm and training data.

Model tuning is a laborious machine learning task that relies heavily on data scientist
expertise. With limited user input, the `oml.automl.ModelTuning` class automates this process
using a highly-parallel, asynchronous gradient-based hyperparameter optimization algorithm
to tune the hyperparameters of an Oracle Machine Learning algorithm.

The `oml.automl.ModelTuning` class supports classification and regression algorithms. To use
the `oml.automl.ModelTuning` class, you specify a data set and an algorithm to obtain a tuned
model and its corresponding hyperparameters. An advanced user can provide a customized hyperparameter search space and a non-default scoring metric to this black box optimizer.

For a partitioned model, if you pass in the column to partition on in the `param_space` argument of the `tune` method, `oml.automl.ModelTuning` tunes the partitioned model’s hyperparameters.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelTuning)` or see Oracle Machine Learning for Python API Reference.

Example 6-3 Using the `oml.automl.ModelTuning` Class

This example creates an `oml.automl.ModelTuning` object.

```python
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
b_data = bc.data.astype(float)
X = pd.DataFrame(b_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
omm_df = oml.create(pd.concat([X, y], axis=1),
                     table = 'BreastCancer')

# Split the data set into training and test data.
train, test = omm_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Start an automated model tuning run with a Decision Tree model.
at = automl.ModelTuning(mining_function='classification',
                        parallel=4)
results = at.tune('dt', X, y, score_metric='accuracy')

# Show the tuned model details.
tuned_model = results['best_model']
tuned_model

# Show the best tuned model train score and the
# corresponding hyperparameters.
score, params = results['all_evals'][0]
"{:.2}".format(score), "{"{:}"}".format(k, params[k])
    for k in sorted(params)

# Use the tuned model to get the score on the test set.
"{:.2}".format(tuned_model.score(X_test, y_test))

# An example invocation of model tuning with user-defined
# search ranges for selected hyperparameters on a new tuning
# metric (f1_macro).

define search space:

- RFOR_SAMPLING_RATIO: type='continuous', range=[0.01, 0.5]
- RFOR_NUM_TREES: type='discrete', range=[50, 100]
- TREE_IMPURITY_METRIC: type='categorical', range=[TREE_IMPURITY_ENTROPY, TREE_IMPURITY_GINI]

results = at.tune('rf', X, y, score_metric='f1_macro', param_space=search_space)

score, params = results['all_evals'][0]

for k in sorted(params):
    print(f'\{k}: {params[k]}')

# Some hyperparameter search ranges need to be defined based on the
# training data set sizes (for example, the number of samples and
# features). You can use placeholders specific to the data set,
# such as $nr_features and $nr_samples, as the search ranges.

search_space = {'RFOR_MTRY': type='discrete', range=[1, $nr_features/2]}

results = at.tune('rf', X, y, score_metric='f1_macro', param_space=search_space)

score, params = results['all_evals'][0]

for k in sorted(params):
    print(f'\{k}: {params[k]}')

# Drop the database table.

oml.drop('BreastCancer')

Listing for This Example

```python
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> X = pd.DataFrame(bc.data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])

>>> # Create the database table BreastCancer.
... oml_df = oml.create(pd.concat([X, y], axis=1),
                        table = 'BreastCancer')

>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> # Start an automated model tuning run with a Decision Tree model.
... at = automl.ModelTuning(mining_function='classification',
```
... parallel=4)
>>> results = at.tune('dt', X, y, score_metric='accuracy')

>>> 

# Show the tuned model details.
... tuned_model = results['best_model']

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: TARGET

Settings:

<table>
<thead>
<tr>
<th>setting name</th>
<th>setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALGO_NAME</td>
</tr>
<tr>
<td></td>
<td>ALGO_DECISION_TREE</td>
</tr>
<tr>
<td>1</td>
<td>CLAS_MAX_SUP_BINS</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>CLAS_WEIGHTS_BALANCED</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
</tr>
<tr>
<td>3</td>
<td>ODMS_DETAILS</td>
</tr>
<tr>
<td></td>
<td>ODMS_DISABLE</td>
</tr>
<tr>
<td>4</td>
<td>ODMS_MISSING_VALUE_TREATMENT</td>
</tr>
<tr>
<td></td>
<td>ODMS_MISSING_VALUE_AUTO</td>
</tr>
<tr>
<td>5</td>
<td>ODMS_SAMPLING</td>
</tr>
<tr>
<td></td>
<td>ODMS_SAMPLING_DISABLE</td>
</tr>
<tr>
<td>6</td>
<td>PREP_AUTO</td>
</tr>
<tr>
<td></td>
<td>ON</td>
</tr>
<tr>
<td>7</td>
<td>TREE_IMPURITY_METRIC</td>
</tr>
<tr>
<td></td>
<td>TREE_IMPURITY_GINI</td>
</tr>
<tr>
<td>8</td>
<td>TREE_TERM_MAX_DEPTH</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>TREE_TERM_MINPCT_NODE</td>
</tr>
<tr>
<td></td>
<td>3.34</td>
</tr>
<tr>
<td>10</td>
<td>TREE_TERM_MINPCT_SPLIT</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>11</td>
<td>TREE_TERM_MINREC_NODE</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>TREE_TERM_MINREC_SPLIT</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

Attributes:
- mean radius
- mean texture
- mean perimeter
- mean area
- mean smoothness
- mean compactness
- mean concavity
- mean concave points
- mean symmetry
- mean fractal dimension
- radius error
- texture error
- perimeter error
- area error
- smoothness error
- compactness error
- concavity error
- concave points error
- symmetry error
- fractal dimension error
- worst radius
- worst texture
- worst perimeter
- worst area
- worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

Partition: NO

>>> # Show the best tuned model train score and the corresponding hyperparameters.
... score, params = results['all_evals'][0]
... '{:.2}'.format(score), ['{}:{}'.format(k, params[k])
... for k in sorted(params)]
(0.92, ['CLAS_MAX_SUP_BINS:32', 'TREE_IMPURITY_METRIC:TREE_IMPURITY_GINI',
'TREE_TERM_MAX_DEPTH:7', 'TREE_TERM_MINPCT_NODE:0.05',
'TREE_TERM_MINPCT_SPLIT:0.1'])

>>> # Use the tuned model to get the score on the test set.
... '{:.2}'.format(tuned_model.score(X_test, y_test))
0.92

>>> # An example invocation of model tuning with user-defined search ranges for selected hyperparameters on a new tuning metric (f1_macro).
... search_space = {
... 'RFOR_SAMPLING_RATIO': {'type': 'continuous',
... 'range': [0.01, 0.5]},
... 'RFOR_NUM_TREES': {'type': 'discrete',
... 'range': [50, 100]},
... 'TREE_IMPURITY_METRIC': {'type': 'categorical',
... 'range': ['TREE_IMPURITY_ENTROPY',
... 'TREE_IMPURITY_GINI']},
...}
>>> results = at.tune('rf', X, y, score_metric='f1_macro',
... param_space=search_space)
>>> score, params = results['all_evals'][0]
... '{:.2}'.format(score), ['{}:{}'.format(k, params[k])
... for k in sorted(params)]
(0.92, ['RFOR_NUM_TREES:53', 'RFOR_SAMPLING_RATIO:0.4999951',
'TREE_IMPURITY_METRIC:TREE_IMPURITY_ENTROPY'])

>>> # Some hyperparameter search ranges need to be defined based on the training data set sizes (for example, the number of samples and features). You can use placeholders specific to the data set, such as $nr_features and $nr_samples, as the search ranges.
... search_space = {'RFOR_MTRY': {'type': 'discrete',
... 'range': [1, '$nr_features/2']})
>>> results = at.tune('rf', X, y,
... score_metric='f1_macro', param_space=search_space)
>>> score, params = results['all_evals'][0]
... '{:.2}'.format(score), ['{}:{}'.format(k, params[k])
... for k in sorted(params)]
(0.93, ['RFOR_MTRY:10'])
>>> # Drop the database table.
... oml.drop('BreastCancer')

Related Topics

- **About Automated Machine Learning**
  Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.

Model Selection

The `oml.automl.ModelSelection` class automatically selects an Oracle Machine Learning algorithm according to the selected score metric and then tunes that algorithm.

The `oml.automl.ModelSelection` class supports classification and regression algorithms. To use the `oml.automl.ModelSelection` class, you specify a data set and the number of algorithms you want to tune.

The `select` method of the class returns the best model out of the models considered.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelSelection)` or see [Oracle Machine Learning for Python API Reference](#).

**Example 6-4 Using the `oml.automl.ModelSelection` Class**

This example creates an `oml.automl.ModelSelection` object and then uses the object to select and tune the best model.

```python
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bcs = datasets.load_breast_cancer()
bcs_data = bcs.data.astype(float)
X = pd.DataFrame(bcs_data, columns = bcs.feature_names)
y = pd.DataFrame(bcs.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                     table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated model selection object with f1_macro as the # score_metric argument.
ms = automl.ModelSelection(mining_function='classification',
                           score_metric='f1_macro', parallel=4)
```
# Run model selection to get the top (k=1) predicted algorithm
# (defaults to the tuned model).
select_model = ms.select(X, y, k=1)

# Show the selected and tuned model.
select_model

# Score on the selected and tuned model.
"{:.2}".format(select_model.score(X_test, y_test))

# Drop the database table.
oml.drop('BreastCancer')

Listing for This Example

>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets

>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])

>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                      table = 'BreastCancer')

>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']

>>> # Create an automated model selection object with f1_macro as the
... # score_metric argument.
... ms = automl.ModelSelection(mining_function='classification',
...                            score_metric='f1_macro', parallel=4)

>>> # Run the model selection to get the top (k=1) predicted algorithm
... # (defaults to the tuned model).
... select_model = ms.select(X, y, k=1)

>>> # Show the selected and tuned model.
... select_model

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: TARGET

Settings:
setting name   setting value
0     ALGO_NAME   ALGO_SUPPORT_VECTOR_MACHINES
1      CLAS_WEIGHTS_BALANCED   OFF
2     ODMS_Details           ODMS_DISABLE
3  ODMS_MISSING_VALUE_TREATMENT   ODMS_MISSING_VALUE_AUTO
4     ODMS_SAMPLING            ODMS_SAMPLING_DISABLE
5      PREP_AUTO               ON
6     SVMS_COMPLEXITY_FACTOR   10
7     SVMS_CONV_TOLERANCE     .0001
8    SVMS_KERNEL_FUNCTION     SVMS_GAUSSIAN
9     SVMS_NUM_PIVOTS          ...
10    SVMS_STD_DEV           5.3999999999999995

Attributes:
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture
Partition: NO

>>> # Score on the selected and tuned model.
... "{:2f}".format(select_model.score(X_test, y_test))
'0.99'
>>> # Drop the database table.
... oml.drop('BreastCancer')
Related Topics

- **About Automated Machine Learning**
  Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.
Embedded Python Execution

With Embedded Python Execution, you can invoke user-defined Python functions in Python engines spawned and managed by the Oracle database instance.

The following topics describe Embedded Python Execution in Oracle Machine Learning for Python:

• About Embedded Python Execution and the Script Repository
• Run a User-Defined Python Function
• Run a User-Defined Python Function on the Specified Data
• Run a Python Function on Data Grouped By Column Values
• Run a User-Defined Python Function on Sets of Rows
• Run a User-Defined Python Function Multiple Times
• Save and Manage User-Defined Python Functions in the Script Repository

About Embedded Python Execution and the Script Repository

OML4Py Embedded Python Execution provides users the ability to invoke user-defined Python functions in one or more Python engines spawned and managed by the Oracle database environment.

About Embedded Python Execution

You may choose to run your functions in a data-parallel or task-parallel manner in one or more of these Python engines. In data-parallel processing, you partition the data and invoke the same user-defined Python function of each data subset using one or more Python engines. In task-parallel processing, you invoke a user-defined function multiple times in one or more Python engines with a unique index passed in as an argument; for example, you may use task parallelism for Monte Carlo simulations in which you use the index to set a random seed.

Embedded Python Execution is available in the following interfaces:

• In an OML Notebooks Python interpreter session
• The REST API for Embedded Python Execution
• For Oracle Autonomous Database:
  – In an OML Notebooks Python interpreter session
  – The REST API for Embedded Python Execution
• For an on-premises Oracle database:
  – The Embedded Python Execution Python classes
  – The SQL interface for Embedded Python Execution

The following table lists the Python functions for Embedded Python Execution.
### Function Description

**oml.do_eval**
Runs a user-defined Python function in a Python engine spawned and managed by the database environment.

**oml.group_apply**
Partitions a database table by the values in one or more columns and runs the provided user-defined Python function on each partition.

**oml.index_apply**
Runs a Python function multiple times, passing in a unique index of the invocation to the user-defined function.

**oml.row_apply**
Partitions a database table into sets of rows and runs the provided user-defined Python function on the data in each set.

**oml.table_apply**
Runs a Python function on data in the database as a single pandas.DataFrame in a single Python engine.

For the SQL interface for Embedded Python Execution, see [SQL for Embedded Python Execution](#).

### About Special Control Arguments

Special control arguments control what happens before or after the running of the function that you pass to an Embedded Python Execution function. You specify a special control argument with the `**kwargs` parameter of a function such as `oml.do_eval`. The control arguments are not passed to the function specified by the `func` argument of that function.

**Table 7-1  Special Control Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oml_input_type</code></td>
<td>Identifies the type of input data object that you are supplying to the <code>func</code> argument. The input types are the following:</td>
</tr>
<tr>
<td></td>
<td>• pandas.DataFrame</td>
</tr>
<tr>
<td></td>
<td>• numpy.recarray</td>
</tr>
<tr>
<td></td>
<td>• 'default' (the default value)</td>
</tr>
<tr>
<td></td>
<td>If all columns are numeric, then default type is a 2-dimensional</td>
</tr>
<tr>
<td></td>
<td>numpy.ndarray of type numpy.float64. Otherwise, the default type is</td>
</tr>
<tr>
<td></td>
<td>a pandas.DataFrame.</td>
</tr>
</tbody>
</table>

| `oml_na_omit`     | Controls the handling of missing values in the input data. If you specify  |
|                   | `oml_na_omit = True`, then rows that contain missing values are removed   |
|                   | from the input data. If all of the rows contain missing values, then the output data is an empty pandas.DataFrame. The default value is False. |

### About Output

When a user-defined Python function runs in OML4Py, by default it returns the Python objects returned by the function. Also, OML4Py captures all `matplotlib.figure.Figure` objects created by the user-defined Python function and converts them into PNG format.

If `graphics = True`, the Embedded Python Execution functions return `oml.embed.data_image._DataImage` objects. The `oml.embed.data_image._DataImage`
class contains Python objects and PNG images. Calling the method __repr__() displays the PNG images and prints out the Python object. By default, .dat returns the Python object that the user-defined Python function returned; .img returns a list containing PNG image data for each figure.

About the Script Repository

Embedded Python Execution includes the ability to create and store user-defined Python functions in the OML4Py script repository, grant or revoke the read privilege to a user-defined Python function, list the available user-defined Python functions, load user-defined Python functions into the Python environment, or drop a user-defined Python function from the script repository.

Along with whatever other actions a user-defined Python function performs, it can also create, retrieve, and modify Python objects that are stored in OML4Py datastores.

In Embedded Python Execution, a user-defined Python function runs in one or more Python engines spawned and managed by the database environment. The engines are dynamically started and managed by the database. From the same user-defined Python function you can get structured data and PNG images.

You can make the user-defined Python function either private or global. A global function is available to any user. A private function is available only to the owner or to users to whom the owner of the function has granted the read privilege.

Run a User-Defined Python Function

Use the oml.do_eval function to run a user-defined input function that explicitly retrieves data or for which external data is not required.

The oml.do_eval function runs a user-defined Python function in a Python engine spawned and managed by the database environment.

The syntax of the oml.do_eval function is the following:

\[
\text{oml.do_eval}(\text{func, func_owner=None, graphics=False, **kwargs})
\]

The \text{func} argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An \text{oml.script.script.Callable} object returned by the \text{oml.script.load} function

The optional \text{func_owner} argument is a string or \text{None} (the default) that specifies the owner of the registered user-defined Python function when argument \text{func} is a registered user-defined Python function name.

The \text{graphics} argument is a boolean that specifies whether to look for images. The default value is \text{False}.

With the **kwargs parameter, you can pass additional arguments to the \text{func} function. Special control arguments, which start with oml_, are not passed to the function specified by \text{func}, but instead control what happens before or after the running of the function.
See Also: About Special Control Arguments

The `oml.do_eval` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.do_eval` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

See Also: About Output

Example 7-1 Using the `oml.do_eval` Function

This example defines a Python function that returns a Pandas DataFrame with the columns ID and RES. It then passes that function to the `oml.do_eval` function.

```python
import pandas as pd
import oml

def return_df(num, scale):
    import pandas as pd
    id = list(range(0, int(num)))
    res = [i/scale for i in id]
    return pd.DataFrame({"ID":id, "RES":res})

res = oml.do_eval(func=return_df, scale = 100, num = 10)
type(res)
res
```

Listing for This Example

```bash
>>> import pandas as pd
>>> import oml

>>> def return_df(num, scale):
...   import pandas as pd
...   id = list(range(0, int(num)))
...   res = [i/scale for i in id]
...   return pd.DataFrame({"ID":id, "RES":res})
... >>>

>>> res = oml.do_eval(func=return_df, scale = 100, num = 10)
>>> type(res)
<class 'pandas.core.frame.DataFrame'>

>>> res
    ID  RES
0  0.0  0.00
1  1.0  0.01
2  2.0  0.02
3  3.0  0.03
4  4.0  0.04
5  5.0  0.05
6  6.0  0.06
7  7.0  0.07
```
Run a User-Defined Python Function on the Specified Data

Use the `oml.table_apply` function to run a Python function on data that you specify with the `data` parameter.

The `oml.table_apply` function runs a user-defined Python function in a Python engine spawned and managed by the database environment. With the `func` parameter, you can supply a Python function or you can specify the name of a user-defined Python function in the OML4Py script repository.

The syntax of the function is the following:

```python
oml.table_apply(data, func, func_owner=None, graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the data that the `func` function operates on.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the execution of the function.

See Also: About Special Control Arguments

The `oml.table_apply` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.table_apply` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

See Also: About Output
Example 7-2   Using the oml.table_apply Function

This example builds a regression model using in-memory data, and then uses the
oml.table_apply function to predict using the model on the first 10 rows of the IRIS
table.

```python
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])

y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
    2: 'virginica'}[x], iris.target)),
    columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model using in-memory data.
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
    iris[['Sepal_Length']])
regr.coef_

# Use oml.table_apply to predict using the model on the first 10
# rows of the IRIS table.
def predict(dat, regr):
    import pandas as pd
    pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
        'Petal_Width']])
    return pd.concat([dat,pd.DataFrame(pred)], axis=1)

res = oml.table_apply(data=oml_iris.head(n=10),
    func=predict, regr=regr)
res
```

Listing for This Example

```python
>>> import oml
>>> import pandas as pd
```
>>> from sklearn import datasets
>>> from sklearn import linear_model

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()

>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])

>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass

>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

>>> # Build a regression model using in-memory data.
... iris = oml_iris.pull()

>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
...          iris[['Sepal_Length']])

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

>>> regr.coef_
array([[ 0.65083716,  0.70913196, -0.55648266]])

>>> # Use oml.table_apply to predict using the model on the first 10
... # rows of the IRIS table.
... def predict(dat, regr):
...     import pandas as pd
...     pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
...                             'Petal_Width']])
...     return pd.concat([dat,pd.DataFrame(pred)], axis=1)

... res = oml.table_apply(data=oml_iris.head(n=10),
...                        func=predict, regr=regr)

<table>
<thead>
<tr>
<th></th>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Petal_Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.6</td>
<td>3.6</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>5.1</td>
<td>2.5</td>
<td>3</td>
<td>1.1</td>
</tr>
<tr>
<td>2</td>
<td>6.0</td>
<td>2.2</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>5.8</td>
<td>2.6</td>
<td>4</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>5.5</td>
<td>2.3</td>
<td>4</td>
<td>1.3</td>
</tr>
<tr>
<td>5</td>
<td>5.5</td>
<td>2.5</td>
<td>4</td>
<td>1.3</td>
</tr>
<tr>
<td>6</td>
<td>6.1</td>
<td>2.8</td>
<td>4</td>
<td>1.3</td>
</tr>
<tr>
<td>7</td>
<td>5.7</td>
<td>2.5</td>
<td>5</td>
<td>2.0</td>
</tr>
<tr>
<td>8</td>
<td>6.0</td>
<td>2.2</td>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>9</td>
<td>6.3</td>
<td>2.5</td>
<td>5</td>
<td>1.9</td>
</tr>
</tbody>
</table>
Run a Python Function on Data Grouped By Column Values

Use the oml.group_apply function to group the values in a database table by one or more columns and then run a user-defined Python function on each group.

The oml.group_apply function runs a user-defined Python function in a Python engine spawned and managed by the database environment. The oml.group_apply function passes the oml.DataFrame specified by the data argument to the user-defined func function as its first argument. The index argument to oml.group_apply specifies the columns of the oml.DataFrame by which the database groups the data for processing by the user-defined Python function. The oml.group_apply function can use data-parallel execution, in which one or more Python engines perform the same Python function on different groups of data.

The syntax of the function is the following.

```python
oml.group_apply(data, index, func, func_owner=None, parallel=None, orderby=None, graphics=False, **kwargs)
```

The data argument is an oml.DataFrame that contains the in-database data that the func function operates on.

The index argument is an oml.DataFrame object, the columns of which are used to group the data before sending it to the func function.

The func argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An oml.script.script.Callable object returned by the oml.script.load function
The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

The optional `orderby` argument is an `oml.DataFrame`, `oml.Float`, or `oml.String` that specifies the ordering of the group partitions.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

See Also: About Special Control Arguments

The `oml.group_apply` function returns a `dict` of Python objects or a `dict` of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.group_apply` returns a `dict` of Python object returned by the function. Otherwise, it returns a `dict` of `oml.embed.data_image._DataImage` objects.

See Also: About Output

Example 7-3  Using the `oml.group_apply` Function

This example defines some functions and calls `oml.group_apply` for each function.

```python
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                             'Petal_Length', 'Petal_Width'])

y = pd.DataFrame(list(map(lambda x:
                           {0: 'setosa', 1: 'versicolor',
                            2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
```
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Define a function that counts the number of rows and returns a
# dataframe with the species and the count.
def group_count(dat):
    import pandas as pd
    return pd.DataFrame([(dat["Species"])[0], dat.shape[0]],
                        columns = ["Species", "COUNT"])

# Select the Species column to use as the index argument.
index = oml.DataFrame(oml_iris["Species"])

# Group the data by the Species column and run the user-defined
# function for each species.
res = oml.group_apply(oml_iris, index, func=group_count,
                      oml_input_type="pandas.DataFrame")

res

# Define a function that builds a linear regression model, with
# Petal_Width as the feature and Petal_Length as the target value,
# and that returns the model after fitting the values.
def build_lm(dat):
    from sklearn import linear_model
    lm = linear_model.LinearRegression()
    X = dat["Petal_Width"]
    y = dat["Petal_Length"]
    lm.fit(X, y)
    return lm

# Run the model for each species and return an objectList in
# dict format with a model for each species.
mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
                                "Species"]], index, func=build_lm)

# The output is a dict of key-value pairs for each species and model.
type(mod)

# Sort dict by the key species.
{k: mod[k] for k in sorted(mod.keys())}

Listing for This Example

>>> import pandas as pd
>>> from sklearn import datasets
>>> import oml

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()

>>> x = pd.DataFrame(iris.data,
...                   columns = ['Sepal_Length','Sepal_Width',
...                              'Petal_Length','Petal_Width'])

>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             "Species": x[4],
...                             })))

Chapter 7
Run a Python Function on Data Grouped By Column Values

7-10
Run a Python Function on Data Grouped By Column Values

>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass

>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table='IRIS')

>>> # Define a function that counts the number of rows and returns a
... # dataframe with the species and the count.
... def group_count(dat):
...     import pandas as pd
...     return pd.DataFrame((dat["Species"][0], dat.shape[0]),
...                         columns="Species", "COUNT")

>>> # Select the Species column to use as the index argument.
... index = oml.DataFrame(oml_iris['Species'])

>>> # Group the data by the Species column and run the user-defined
... # function for each species.
... res = oml.group_apply(oml_iris, index, func=group_count,
...                       oml_input_type="pandas.DataFrame")

>>> res
{"setosa": Species COUNT
  0 setosa     50, "versicolor": Species COUNT
  0 versicolor     50, "virginica": Species COUNT
  0 virginica     50}

>>> # Define a function that builds a linear regression model, with
... # Petal_Width as the feature and Petal_Length as the target value,
... # and that returns the model after fitting the values.
... def build_lm(dat):
...     from sklearn import linear_model
...     lm = linear_model.LinearRegression()
...     X = dat["Petal_Width"]
...     y = dat["Petal_Length"]
...     lm.fit(X, y)
...     return lm

>>> # Run the model for each species and return an objectList in
... # dict format with a model for each species.
... mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
...                                    "Species"]], index, func=build_lm)

>>> # The output is a dict of key-value pairs for each species and model.
... type(mod)
<class 'dict'>

>>> # Sort dict by the key species.
... {k: mod[k] for k in sorted(mod.keys())}
{"setosa": LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=None, normalize=False), 'versicolor':
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False), 'virginica': LinearRegression(copy_X=True,
fit_intercept=True, n_jobs=None, normalize=False)}

Related Topics

• Run a User-Defined Python Function on Sets of Rows
  Use the oml.row_apply function to chunk data into sets of rows and then run a
  user-defined Python function on each chunk.

Run a User-Defined Python Function on Sets of Rows

Use the oml.row_apply function to chunk data into sets of rows and then run a user-defined Python function on each chunk.

The oml.row_apply function passes the oml.DataFrame specified by the data
argument as the first argument to the user-defined func Python function. The rows
argument specifies the maximum number of rows of the oml.DataFrame to assign to
each chunk. The last chunk of rows may have fewer rows than the number specified.

The oml.row_apply function runs the Python function in a database-spawned Python
engine. The function can use data-parallel execution, in which one or more Python
engines perform the same Python function on different chunks of the data.

The syntax of the function is the following.

oml.row_apply(data, func, func_owner=None, rows=1, parallel=None,
graphics=False, **kwargs)

The data argument is an oml.DataFrame that contains the data that the func function
operates on.

The func argument is the function to run. It may be one of the following:

• A Python function
• A string that is the name of a user-defined Python function in the OML4Py script
  repository
• A string that defines a Python function
• An oml.script.script.Callable object returned by the oml.script.load
  function

The optional func_owner argument is a string or None (the default) that specifies the
owner of the registered user-defined Python function when argument func is a
registered user-defined Python function name.

The rows argument is an int that specifies the maximum number of rows to include in
each chunk.

The parallel argument is a boolean, an int, or None (the default) that specifies the
preferred degree of parallelism to use in the Embedded Python Execution job. The
value may be one of the following:

• A positive integer greater than or equal to 1 for a specific degree of parallelism
• False, None, or 0 for no parallelism
• True for the default data parallelism

The graphics argument is a boolean that specifies whether to look for images. The default value is True.

With the **kwargs parameter, you can pass additional arguments to the func function. Special control arguments, which start with oml_, are not passed to the function specified by func, but instead control what happens before or after the running of the function.

See Also: About Special Control Arguments

The oml.row_apply function returns a pandas.DataFrame or a list of oml.embed.data_image._DataImage objects. If no image is rendered in the user-defined Python function, oml.row_apply returns a pandas.DataFrame. Otherwise, it returns a list of oml.embed.data_image._DataImage objects.

See Also: About Output

Example 7-4 Using the oml.row_apply Function

This example creates the x and y variables using the iris data set. It then creates the persistent database table IRIS and the oml.DataFrame object oml_iris as a proxy for the table.

The example builds a regression model based on iris data. It defines a function that predicts the Petal_Width values based on the Sepal_Length, Sepal_Width, and Petal_Length columns of the input data. It then concatenates the Species column, the Petal_Width column, and the predicted Petal_Width as the object to return. Finally, the example calls the oml.row_apply function to apply the make_pred() function on each 4-row chunk of the input data.

```python
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                            'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                            {0: 'setosa', 1: 'versicolor',
                             2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model to predict Petal_Width using in-memory data.
```
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Length', 'Sepal_Width', 'Petal_Length']],
         iris[['Petal_Width']])
regr.coef_

# Define a Python function.
def make_pred(dat, regr):
    import pandas as pd
    import numpy as np
    pred = regr.predict(dat[['Sepal_Length',
                              'Sepal_Width',
                              'Petal_Length']])
    return pd.concat([dat[['Species', 'Petal_Width']],
                      pd.DataFrame(pred,
                          columns=['Pred_Petal_Width']),
                      axis=1])

input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
input_data.crosstab(index='Species').sort_values('Species')

res = oml.row_apply(input_data, rows=4, func=make_pred,
                     regr=regr, parallel=2)
type(res)
res

Listing for This Example

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length',
                              'Sepal_Width',
                              'Petal_Length',
                              'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                             {0: 'setosa', 1: 'versicolor',
                              2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass

>>> # Create the IRIS database table and the proxy object for the table.
>>> oml_iris = oml.create(pd.concat([x, y], axis=1), table='IRIS')

>>> # Build a regression model to predict Petal_Width using in-memory
... # data.
... iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[[©Sepal_Length©, ©Sepal_Width©, ©Petal_Length©]],
...           iris[[©Petal_Width©]])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
>>> regr.coef_
array([[-0.20726607,  0.22282854,  0.52408311]])

>>> # Define a Python function.
... def make_pred(dat, regr):
...     import pandas as pd
...     import numpy as np
...     pred = regr.predict(dat[[©Sepal_Length©,
...                              ©Sepal_Width©,
...                              ©Petal_Length©]])
...     return pd.concat([dat[[©Species©, ©Petal_Width©]],
...                        pd.DataFrame(pred,
...                                     columns=[©Pred_Petal_Width©])],
...                      axis=1)

input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
>>> input_data.crosstab(index = ©Species©).sort_values(©Species©)
SPECIES  count
0      setosa      7
1  versicolor      8
2   virginica      4

>>> res = oml.row_apply(input_data, rows=4, func=make_pred, regr=regr,
...                      columns=[©Species©,
...                              ©Petal_Width©,
...                              ©Pred_Petal_Width©])
>>> type(res)
<class 'pandas.core.frame.DataFrame'>
>>> res
     Species  Petal_Width  Pred_Petal_Width
0      setosa 0.4          0.344846
1      setosa 0.3          0.335509
2      setosa 0.2          0.294117
3      setosa 0.2          0.220982
4      setosa 0.2          0.080937
5    versicolor 1.5        1.504615
6    versicolor 1.3        1.560570
7    versicolor 1.0        1.008352
8    versicolor 1.3        1.131905
9    versicolor 1.3        1.215622
10   versicolor 1.3        1.272388
11   virginica 1.8        1.623561
12   virginica 1.8        1.878132
Run a User-Defined Python Function Multiple Times

Use the `oml.index_apply` function to run a Python function multiple times in Python engines spawned by the database environment.

The syntax of the function is the following:

```python
oml.index_apply(times, func, func_owner=None, parallel=None, graphics=False, **kwargs)
```

The `times` argument is an `int` that specifies the number of times to run the `func` function.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, None, or 0 for no parallelism
- `True` for the default data parallelism

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `True`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

**See Also:** About Special Control Arguments

The `oml.index_apply` function returns a list of Python objects or a list of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.index_apply` returns a list of the Python objects returned by the user-defined Python function. Otherwise, it returns a list of `oml.embed.data_image._DataImage` objects.
See Also: About Output

Example 7-5  Using the oml.index_apply Function

This example defines a function that returns the mean of a set of random numbers the specified number of times.

```python
import oml
import pandas as pd

def compute_random_mean(index):
    import numpy as np
    import scipy
    from statistics import mean
    np.random.seed(index)
    res = np.random.random((100,1))*10
    return mean(res[1])
res = oml.index_apply(times=10, func=compute_random_mean)
type(res)
res
```

Listing for This Example

```python
>>> import oml
>>> import pandas as pd

>>> def compute_random_mean(index):
...     import numpy as np
...     import scipy
...     from statistics import mean
...     np.random.seed(index)
...     res = np.random.random((100,1))*10
...     return mean(res[1])
...

>>> res = oml.index_apply(times=10, func=compute_random_mean)
>>> type(res)
<class 'list'>
>>> res
[7.203244934421581, 0.25926231827891333, 7.081478226181048, 5.4723224917572235, 8.707323061773764, 3.3197980530117723, 7.7991879224011464, 9.68540662820932, 5.018745921487388, 0.207519493594015]
```

Save and Manage User-Defined Python Functions in the Script Repository

The OML4Py script repository stores user-defined Python functions for use with Embedded Python Execution functions.

The script repository is a component of the Embedded Python Execution functionality.

The following topics describe the script repository and the Python functions for managing user-defined Python functions:
• About the Script Repository
• Create and Store a User-Defined Python Function
• List Available User-Defined Python Functions
• Load a User-Defined Python Function
• Drop a User-Defined Python Function from the Repository

About the Script Repository

Use these functions to store, manage, and use user-defined Python functions in the script repository.

The following table lists the Python functions for the script repository.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.script.create</td>
<td>Registers a single user-defined Python function in the script repository.</td>
</tr>
<tr>
<td>oml.script.dir</td>
<td>Lists the user-defined Python functions present in the script repository.</td>
</tr>
<tr>
<td>oml.script.drop</td>
<td>Drops a user-defined Python function from the script repository.</td>
</tr>
<tr>
<td>oml.script.load</td>
<td>Loads a user-defined Python function from the script repository into a Python session.</td>
</tr>
</tbody>
</table>

The following table lists the Python functions for managing access to user-defined Python functions in the script repository, and to datastores and datastore objects.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oml.grant</td>
<td>Grants read privilege permission to another user to a datastore or user-defined Python function owned by the current user.</td>
</tr>
<tr>
<td>oml.revoke</td>
<td>Revokes the read privilege permission that was granted to another user to a datastore or user-defined Python function owned by the current user.</td>
</tr>
</tbody>
</table>

Related Topics
• Manage Access to Stored Objects
  The oml.grant and oml.revoke functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

Create and Store a User-Defined Python Function

Use the oml.script.create function to add a user-defined Python function to the script repository.

With the oml.script.create function, you can store a single user-defined Python function in the OML4Py script repository. You can then specify the user-defined Python function as the func argument to the Embedded Python Execution functions oml.do_eval, oml.group_apply, oml.index_apply, oml.row_apply, and oml.table_apply.

You can make the user-defined Python function either private or global. A private user-defined Python function is available only to the owner, unless the owner grants the
read privilege to other users. A global user-defined Python function is available to any user.

The syntax of `oml.script.create` is the following:

```
oml.script.create(name, func, is_global=False, overwrite=False)
```

The `name` argument is a string that specifies a name for the user-defined Python function in the Python script repository.

The `func` argument is the Python function to run. The argument can be a Python function or a string that contains the definition of a Python function. You must specify a string in an interactive session if `readline` cannot get the command history.

The `is_global` argument is a boolean that specifies whether to create a global user-defined Python function. The default value is `False`, which indicates that the user-defined Python function is a private function available only to the current session user. When `is_global` is `True`, it specifies that the function is global and every user has the read privilege and the execute privilege to it.

The `overwrite` argument is a boolean that specifies whether to overwrite the user-defined Python function if it already exists. The default value is `False`.

**Example 7-6  Using the `oml.script.create` Function**

This example stores two user-defined Python functions in the script repository. It then lists the contents of the script repository using different arguments to the `oml.script.dir` function.

```python
from sklearn import datasets
import pandas as pd
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

# Create objects containing data for the user-defined functions to use.
x = pd.DataFrame(iris.data,
    columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])

y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
                                           2: 'virginica'}[x], iris.target)),
    columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table='IRIS')

def build_lm1(dat):
    from sklearn import linear_model
    regr = linear_model.LinearRegression()
    import pandas as pd
    import pandas as pd
    dat = pd.get_dummies(dat, drop_first=True)
    X = dat[['Sepal_Width', 'Petal_Length', 'Petal_Width',
                     'Species_versicolor', 'Species_virginica']]  
y = dat[['Sepal_Length']]```
regr.fit(X, y)
return regr

# Create a private user-defined Python function.
oml.script.create("MYLM", func=build_lm1)

# Run the user-defined Python function.
res = oml.table_apply(oml_iris, func="MYLM",
    oml_input_type="pandas.DataFrame")
res
res.pull().coef_

# Define another function.
def build_lm2(dat):
    from sklearn import linear_model
    regr = linear_model.LinearRegression()
    X = dat["Petal_Width"]
    y = dat["Petal_Length"]
    regr.fit(X, y)
    return regr

# Create a global user-defined Python function, available to any user.
oml.script.create("GLBLM", func=build_lm2, is_global=True)

# Run the user-defined Python function.
res = oml.table_apply(oml_iris, func="GLBLM",
    oml_input_type="pandas.DataFrame")
res

# Define the same function, specified as a string.
func_str = '''
def build_lm2(dat):
    from sklearn import linear_model
    regr = linear_model.LinearRegression()
    X = dat["Petal_Width"]
    y = dat["Petal_Length"]
    regr.fit(X, y)
    return regr
'''

# Overwrite the previous GLBLM user-defined Python function
# in the script repository.
oml.script.create("GLBLM", func=build_lm2, is_global=True,
    overwrite=True)
res = oml.table_apply(oml_iris, func="GLBLM",
    oml_input_type="pandas.DataFrame")
res

# List the user-defined Python functions in the script repository
# available to the current user only.
oml.script.dir()

# List all of the user-defined Python functions available to the
# current user.
oml.script.dir(sctype='all')
# List the user-defined Python functions available to all users.

```python
oml.script.dir(sctype='global')
```

# Drop the IRIS database table.

```python
oml.drop('IRIS')
```

**Listing for This Example**

```python
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model

>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()

>>> # Create objects containing data for the user-defined Python functions to use.
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])

>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])

>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table='IRIS')

>>> # Define a function.
... def build_lm1(dat):
...   from sklearn import linear_model
...   regr = linear_model.LinearRegression()
...   import pandas as pd
...   dat = pd.get_dummies(dat, drop_first=True)
...   X = dat[['Sepal_Width', 'Petal_Length', 'Petal_Width',
...             'Species_versicolor', 'Species_virginica']]
...   y = dat[['Sepal_Length']]
...   regr.fit(X, y)
...   return regr

>>> # Create a private user-defined Python function.
... oml.script.create("MYLM", func=build_lm1)

>>> # Run the user-defined Python function.
... res = oml.table_apply(oml_iris, func="MYLM",
...                       oml_input_type="pandas.DataFrame")
... res

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

>>> res.pull().coef_  
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])

>>> # Define another function.
... def build_lm2(dat):
...   from sklearn import linear_model
...   regr = linear_model.LinearRegression()
...   X = dat["Petal_Width"]
...   y = dat["Petal_Length"]
...   regr.fit(X, y)
...   return regr
...

>>> # Create a global user-defined Python function available to any user.
... oml.script.create("GLBLM", func=build_lm2, is_global=True)

>>> # Run the user-defined Python function.
... res = oml.table_apply(oml_iris, func="GLBLM",
...   oml_input_type="pandas.DataFrame")
>>> res
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
    normalize=False)

>>> # Define the same function, specified as a string.
... func_str = "def build_lm2(dat):
...   from sklearn import linear_model
...   regr = linear_model.LinearRegression
...   X = dat["Petal_Width"]
...   y = dat["Petal_Length"]
...   regr.fit(X, y)
...   return regr"
...

>>> # Overwrite the previous GLBLM user-defined Python function
>>> # in the script repository.
... oml.script.create("GLBLM", func=build_lm2, is_global=True,
...   overwrite=True)
>>> res = oml.table_apply(oml_iris, func="GLBLM",
...   oml_input_type="pandas.DataFrame")
>>> res
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
    normalize=False)

>>> # List the user-defined Python functions available to all users.
... oml.script.dir(sctype="global")
NAME       SCRIPT
0  GLBLM  def build_lm2(dat):

7-22
List Available User-Defined Python Functions

Use the `oml.script.dir` function to list the user-defined Python functions in the OML4Py script repository.

The syntax of the `oml.script.dir` function is the following:

```python
oml.script.dir(name=None, regex_match=False, sctype='user')
```

The `name` argument is a string that specifies the name of a user-defined Python function or a regular expression to match to the names of user-defined Python functions in the script repository. When `name` is `None`, this function returns the type of user-defined Python functions specified by argument `sctype`.

The `regex_match` argument is a boolean that indicates whether argument `name` is a regular expression to match. The default value is `False`.

The `sctype` argument is a string that specifies the type of user-defined Python function to list. The value may be one of the following.

- `user`, to specify the user-defined Python functions available to the current user only.
- `grant`, to specify the user-defined Python functions to which the read and execute privilege have been granted by the current user to other users.
- `granted`, to specify the user-defined Python functions to which the read and execute privilege have been granted by other users to the current user.
- `global`, to specify all of the global user-defined Python functions created by the current user.
- `all`, to specify all of the user-defined Python functions available to the current user.

The `oml.script.dir` function returns a `pandas.DataFrame` that contains the columns `NAME` and `SCRIPT` and, optionally, the columns `OWNER` and `GRANTEE`.

**Example 7-7 Using the `oml.script.dir` Function**

This example lists the contents of the script repository using different arguments to the `oml.script.dir` function. For the creation of the user-defined Python functions, see Example 7-6.

```python
import oml

# List the user-defined Python functions in the script # repository available to the current user only.
.oml_script.dir()

# List all of the user-defined Python functions available # to the current user.
.oml_script.dir(sctype='all')
```
# List the user-defined Python functions available to all users.
%python
oml.script.dir(sctype='global')

# List the user-defined Python functions that contain the letters
# BL and that are available to all users.
%python
oml.script.dir(name="BL", regex_match=True, sctype='all')

Listing for This Example

```python
>>> import oml

>>> # List the user-defined Python functions in the script
... # repository available to the current user only.
... oml.script.dir()

NAME                                  SCRIPT
0  MYLM  def build_lm1(dat):
    from sklearn import l...

>>> # List all of the user-defined Python functions available
... to the current user.
... oml.script.dir(sctype='all')

OWNER   NAME                                  SCRIPT
0    PYQSYS  GLBLM  def build_lm2(dat):
    from sklearn import l...
1  OML_USER   MYLM  def build_lm1(dat):
    from sklearn import l...

>>> # List the user-defined Python functions available to all users.
>>> oml.script.dir(sctype='global')

NAME                                  SCRIPT
0  GLBLM  def build_lm2(dat):
    from sklearn import l...

>>> # List the user-defined Python functions that contain the letters
... # BL and that are available to all users.
... oml.script.dir(name="BL", regex_match=True, sctype='all')

OWNER   NAME                                  SCRIPT
0  PYQSYS  GLBLM  def build_lm2(dat):
    from sklearn import l...
```

Load a User-Defined Python Function

Use the `oml.script.load` function to load a user-defined Python function from the script repository into a Python session.

The syntax of the function is the following:

```python
oml.script.load(name\, owner=None)
```

The `name` argument is a string that specifies the name of the user-defined Python function to load from the OML4Py script repository.

The optional `owner` argument is a string that specifies the owner of the user-defined Python function or `None` (the default). If `owner=None`, then this function finds and loads the user-defined Python function that matches `name` in the following order:

1. A user-defined Python function that the current user created.
2. A global user-defined Python function that was created by another user.
The `oml.script.load` function returns an `oml.script.script.Callable` object that references the named user-defined Python function.

**Example 7-8 Using the `oml.script.load` Function**

This example loads user-defined Python functions from the script repository and pulls them to the local Python session. For the creation of the user-defined Python functions, see Example 7-6.

```python
import oml

# Load the MYLM and GLBLM user-defined Python functions.
MYLM = oml.script.load(name="MYLM")
GMYLM = oml.script.load(name="GLBLM")

# Pull the models to the local Python session.
MYLM(oml_iris.pull()).coef_
GMYLM(oml_iris.pull())
```

**Listing for This Example**

```bash
>>> import oml

>>> # Load the MYLM and GLBLM user-defined Python functions.
... MYLM = oml.script.load(name="MYLM")
... GMYLM = oml.script.load(name="GLBLM")
... MYLM(oml_iris.pull()).coef_
... array([[ 0.49588894, 0.82924391, -0.31515517, -0.72356196, -1.02349781]])

>>> GMYLM(oml_iris.pull())
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

### Drop a User-Defined Python Function from the Repository

Use the `oml.script.drop` function to remove a user-defined Python function from the script repository.

The `oml.script.drop` function drops a user-defined Python function from the OML4Py script repository.

The syntax of the function is the following:

```python
oml.script.drop(name, is_global=False, silent=False)
```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository.

The `is_global` argument is a boolean that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is `False`, which indicates a private user-defined Python function.
The silent argument is a boolean that specifies whether to display an error message when oml.script.drop encounters an error in dropping the specified user-defined Python function. The default value is False.

**Example 7-9   Using the oml.script.drop Function**

This example drops user-defined Python functions the MYLM private user-defined Python function and the GLBLM global user-defined Python function from the script repository. For the creation of the user-defined Python functions, see Example 7-6.

```python
import oml

# List the available user-defined Python functions.
oml.script.dir(sctype="all")

# Drop the private user-defined Python function.
oml.script.drop("MYLM")

# Drop the global user-defined Python function.
oml.script.drop("GLBLM", is_global=True)

# List the available user-defined Python functions again.
oml.script.dir(sctype="all")
```

**Listing for This Example**

```plaintext
>>> import oml

>>> # List the available user-defined Python functions.
... oml.script.dir(sctype="all")

<table>
<thead>
<tr>
<th>OWNER</th>
<th>NAME</th>
<th>SCRIPT</th>
</tr>
</thead>
</table>
| 0       | PYQSYS   | GLBLM def build_lm2(dat):
| 1       | OML_USER | MYLM  def build_lm1(dat):

>>> # Drop the private user-defined Python function.
... oml.script.drop("MYLM")

>>> # Drop the global user-defined Python function.
... oml.script.drop("GLBLM", is_global=True)

>>> # List the available user-defined Python functions again.
... oml.script.dir(sctype="all")
```

Empty DataFrame
Columns: [OWNER, NAME, SCRIPT]
Index: []
OML4Py has SQL interfaces for Embedded Python Execution and for datastore and script repository management.

The following topics describe the OML4Py SQL interfaces for Embedded Python Execution.

- About the SQL Interface for Embedded Python Execution
- pyqEval Function
- pyqTableEval Function
- pyqRowEval Function
- pyqGroupEval Function
- sys.pyqScriptCreate Procedure
- sys.pyqScriptDrop Procedure
- ALL_PYQ_DATASTORE_CONTENTS View
- ALL_PYQ_DATASTORES View
- ALL_PYQ_SCRIPTS View
- USER_PYQ_DATASTORES View
- USER_PYQ_SCRIPTS View

About the SQL Interface for Embedded Python Execution

With the SQL interface, you can run user-defined Python functions in one or more separate Python engines in an Oracle database environment, manage user-defined Python functions in the OML4Py script repository, and control access to and get information about datastores and about user-defined Python functions in the script repository.

You can use the SQL interface for Embedded Python Execution with an on-premises Oracle Database instance.

OML4Py provides the following types of SQL functions, procedures, and views.

- SQL table functions for running user-defined Python functions in one or more database-spawned and managed Python engines; the user-defined Python functions may reference Python objects in OML4Py datastores and use third-party packages installed with the database server machine Python engines.
- PL/SQL procedures for creating and dropping user-defined Python functions in the OML4Py script repository.
- PL/SQL procedures for granting and revoking the read privilege to datastores and the datastore objects in them, and to user-defined Python functions in the OML4Py script repository.
- Database views that contain information about datastores and user-defined Python functions in the OML4Py script repository.
The following table lists the SQL functions for Embedded Python Execution, the PL/SQL procedures for managing datastores and user-defined Python functions, and the Oracle Database views that contain information about the datastores and user-defined Python functions.

<table>
<thead>
<tr>
<th>Function, Procedure, or View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyqEval function</td>
<td>Runs a user-defined Python function on the data passed in.</td>
</tr>
<tr>
<td>pyqGroupEval function</td>
<td>Groups data by one or more columns and runs a user-defined Python function on each group.</td>
</tr>
<tr>
<td>pyqTableEval function</td>
<td>Runs a user-defined Python function on data in the database.</td>
</tr>
<tr>
<td>pyqRowEval function</td>
<td>Runs the specified number of rows in each invocation of the user-defined Python function in parallel processes.</td>
</tr>
<tr>
<td>pyqGrant procedure</td>
<td>Grants the read privilege to another user to a user-defined Python function owned by the current user.</td>
</tr>
<tr>
<td>pyqRevoke procedure</td>
<td>Revokes the read privilege that was granted to another user to a user-defined Python function owned by the current user.</td>
</tr>
<tr>
<td>pyqScriptCreate procedure</td>
<td>Creates a user-defined Python function in the script repository.</td>
</tr>
<tr>
<td>pyqScriptDrop procedure</td>
<td>Drops a user-defined Python function from the script repository.</td>
</tr>
<tr>
<td>ALL_PYQ_DATASTORES view</td>
<td>Contains information about the datastores available to the current user.</td>
</tr>
<tr>
<td>ALL_PYQ_DATASTORE_CONTENTS view</td>
<td>Contains information about the objects in the datastores available to the current user.</td>
</tr>
<tr>
<td>USER_PYQ_DATASTORES view</td>
<td>Contains information about the datastores owned by the current user.</td>
</tr>
<tr>
<td>ALL_PYQSCRIPTS view</td>
<td>Describes the scripts that are available to the current user.</td>
</tr>
<tr>
<td>USER_PYQSCRIPTS view</td>
<td>Describes the user-defined Python functions in the script repository that are owned by the current user.</td>
</tr>
</tbody>
</table>

### pyqEval Function

Runs a user-defined Python function that explicitly retrieves data or for which external data is to be automatically loaded for the function.

You can pass arguments to the Python function with the `PAR_QRY` parameter.

The `pyqEval` function does not automatically receive any data from the database. The Python function generates the data that it uses or it explicitly retrieves it from a data source such as Oracle Database, other databases, or flat files.

The Python function can return a `bool`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_QRY` parameter.
Syntax

\[ \text{pyqEval (} \]
\[ \begin{array}{lll}
\text{PAR_QRY} & \text{VARCHAR2} & \text{IN} \\
\text{OUT_QRY} & \text{VARCHAR2} & \text{IN} \\
\text{EXP_NAM} & \text{VARCHAR2} & \text{IN})
\end{array} \]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR_QRY</td>
<td>A JSON string that contains parameters to pass to the user-defined Python function specified by the EXP_NAM parameter. Special control arguments, which start with oml_, are not passed to the function specified by EXP_NAM, but instead control what happens before or after the invocation of the function. See Also: About Special Control Arguments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUT_QRY</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- A JSON string that specifies the column names and data types of the table returned by pyqEval. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples.</td>
</tr>
<tr>
<td></td>
<td>- The name of an existing table or view to use as a prototype. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. If using a table or view owned by another user, use the format &lt;owner name&gt;.&lt;table/view name&gt;. You must have read access to the specified table or view.</td>
</tr>
<tr>
<td></td>
<td>- The string 'XML!', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.</td>
</tr>
<tr>
<td></td>
<td>- The string 'PNG!', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function in PNG format.</td>
</tr>
</tbody>
</table>

| EXP_NAM   | The name of a user-defined Python function in the OML4Py script repository. |

Returns

Function \text{pyqEval} returns a table that has the structure specified by the OUT_QRY parameter value.

Example 8-1 Using the pyqEval Function

This example defines Python functions and stores them in the OML4Py script repository. It invokes the pyqEval function on the user-defined Python functions.

In a PL/SQL block, create an unnamed Python function that is stored in script repository with the name pyqFun1.

BEGIN
    sys.pyqScriptCreate('pyqFun1', 'func = lambda: "Hello World from a lambda!"',
                         FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
Invoke the `pyqEval` function, which runs the user-defined Python function and returns the results as XML.

```
SELECT name, value
FROM table(pyqEval(
    NULL,
    'XML',
    'pyqFun1'));
```

The output is the following.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;root&gt;</td>
<td>&lt;str&gt;Hello World from a lambda!&lt;/str&gt; &lt;/root&gt;</td>
</tr>
</tbody>
</table>

Drop the user-defined Python function.

```
BEGIN
    sys.pyqScriptDrop('pyqFun1');
END;
/
```

Define a Python function that returns a `numpy.ndarray` that is stored in script repository with the name `pyqFun2`.

```
BEGIN
    sys.pyqScriptCreate('pyqFun2',
        'def return_frame():
            import numpy as np
            import pickle
            z = np.array(
                [[str(x) + "demo" for x in range(10)],
                 [x / 10 for x in range(10)],
                 [x for x in range(10)],
                 [bool(x % 2) for x in range(10)],
                 [pickle.dumps(x) for x in range(10)],
                 ["test" + str(x**2) for x in range(10)]]),
            dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"),
                   ("d", "b"), ("e", "S20"), ("f", "O")])
            return z';
END;
/
```

Invoke the `pyqEval` function, which runs the `pyqFun2` user-defined Python function.

```
SELECT *
FROM table(pyqEval(
    NULL,
    'XML',
    'pyqFun2');
```

The output is the following.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;root&gt;</td>
<td>&lt;str&gt;Hello World from a lambda!&lt;/str&gt; &lt;/root&gt;</td>
</tr>
</tbody>
</table>
The output is the following.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0demo</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80034B002E</td>
<td>test0</td>
</tr>
<tr>
<td>1demo</td>
<td>1.0E-001</td>
<td>1</td>
<td>0</td>
<td>80034B012E</td>
<td>test1</td>
</tr>
<tr>
<td>2demo</td>
<td>2.0E-001</td>
<td>2</td>
<td>0</td>
<td>80034B022E</td>
<td>test4</td>
</tr>
<tr>
<td>3demo</td>
<td>3.0E-001</td>
<td>3</td>
<td>1</td>
<td>80034B032E</td>
<td>test9</td>
</tr>
<tr>
<td>4demo</td>
<td>4.0E-001</td>
<td>4</td>
<td>0</td>
<td>80034B042E</td>
<td>test16</td>
</tr>
<tr>
<td>5demo</td>
<td>5.0E-001</td>
<td>5</td>
<td>1</td>
<td>80034B052E</td>
<td>test25</td>
</tr>
<tr>
<td>6demo</td>
<td>6.0E-001</td>
<td>6</td>
<td>0</td>
<td>80034B062E</td>
<td>test36</td>
</tr>
<tr>
<td>7demo</td>
<td>7.0E-001</td>
<td>7</td>
<td>1</td>
<td>80034B072E</td>
<td>test49</td>
</tr>
<tr>
<td>8demo</td>
<td>8.0E-001</td>
<td>8</td>
<td>0</td>
<td>80034B082E</td>
<td>test64</td>
</tr>
<tr>
<td>9demo</td>
<td>9.0E-001</td>
<td>9</td>
<td>1</td>
<td>80034B092E</td>
<td>test81</td>
</tr>
</tbody>
</table>

10 rows selected.

Drop the user-defined Python function.

```sql
BEGIN
    sys.pyqScriptDrop('pyqFun2');
END;
/
```

**pyqTableEval Function**

Runs a user-defined Python function on data from an Oracle Database table.

You pass data to the Python function with the `INP_QRY` parameter. You can pass arguments to the Python function with the `PAR_QRY` parameter.

The Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a pandas.DataFrame object. You define the form of the returned value with the `OUT_QRY` parameter.

**Syntax**

```sql
pyqTableEval ( INP_QRY VARCHAR2 IN,
                PAR_QRY VARCHAR2 IN,
                OUT_QRY VARCHAR2 IN,
                EXP_NAM VARCHAR2 IN)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_QRY</td>
<td>The name of a table or view that specifies the data to pass to the Python function specified by the EXP_NAM parameter. If using a table or view owned by another user, use the format &lt;owner name&gt;.&lt;table/view name&gt;. You must have read access to the specified table or view.</td>
</tr>
<tr>
<td>PAR_QRY</td>
<td>A JSON string that contains keyword arguments to pass to the user-defined Python function specified by the EXP_NAM parameter. Special control arguments, which start with oml_, are not passed to the function specified by EXP_NAM, but instead control what happens before or after the running of the function. See Also: About Special Control Arguments</td>
</tr>
</tbody>
</table>
| OUT_QRY   | One of the following:  
- A JSON string that specifies the column names and data types of the table returned by pyqTableEval. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples.  
- The name of an existing table or view to use as a prototype. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.  
- The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.  
- The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function in PNG format. |
| EXP_NAM   | The name of a user-defined Python function in the OML4Py script repository. |

Returns

Function pyqTableEval returns a table that has the structure specified by the OUT_QRY parameter value.

Example 8-2  Using the pyqTableEval Function

This example stores a user-defined Python function in the OML4Py script repository with the name create_iris_table. It uses the function to create a database table as the result of a pyqEval function invocation. It creates another user-defined Python function that fits a linear regression model to the input data and saves the model in the OML4Py datastore. The example runs a SQL SELECT statement that invokes the pyqTableEval function, which invokes the function stored in the script repository with the name myLinearRegressionModel.
In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`, overwriting any existing user-defined Python function stored in the script repository with the same name.

The `create_iris_table` function imports and loads the iris data set, creates two pandas.DataFrame objects, and then returns the concatenation of those objects.

```python
BEGIN
    sys.pyqScriptCreate('create_iris_table',
        'def create_iris_table():
            from sklearn.datasets import load_iris
            import pandas as pd
            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",
                "Sepal_Width", "Petal_Length", "Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",
                2: "virginica"}[x], iris.target)),
                columns = ["Species"]
            return pd.concat([y, x], axis=1),
            FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
CREATE TABLE IRIS AS
(_SELECT * FROM pyqEval(
    NULL,
    {"Species":"VARCHAR2(10)","Sepal_Length":"number",
     "Sepal_Width":"number","Petal_Length":"number",
     "Petal_Width":"number"},
    create_iris_table'
));
```

Define the Python function `fit_model` and store it with the name `myLinearRegressionModel` as a private function in the script repository, overwriting any existing user-defined Python function stored with that name.

The `fit_model` function fits a regression model to the input data `dat` and then saves the fitted model as an object specified by the `modelName` argument to the datastore specified by the `datastoreName` argument. The `fit_model` function returns the fitted model in a string format.

By default, Python objects are saved to a new datastore with the specified `datastoreName`. To save an object to an existing datastore, either set the `overwrite` or `append` argument to `True` in the `oml.ds.save` invocation.

```python
BEGIN
    sys.pyqScriptCreate('myLinearRegressionModel',
        'def fit_model(dat, modelName, datastoreName):
            import oml
            from sklearn import linear_model
            regr = linear_model.LinearRegression()
            regr.fit(dat.loc[:, ["Sepal_Length", "Sepal_Width", 
                "Petal_Length"]], dat.loc[:,["Petal_Width"]])
            oml.ds.save(objs={modelName:regr}, name=datastoreName,
            overwrite=True)
            return str(regr),
            FALSE, TRUE);
```
Run a SELECT statement that invokes the `pyqTableEval` function. The `INP_QRY` parameter of the `pyqTableEval` function specifies the IRIS table as the data to pass to the Python function. The `PAR_QRY` parameter specifies the names of the model and datastore to pass to the Python function, and specifies the `oml_connect` control argument to establish an OML4Py connection to the database during the invocation of the user-defined Python function. The `OUT_QRY` parameter specifies returning the value in XML format and the `EXP_NAM` parameter specifies the `myLinearRegressionModel` function in the script repository as the Python function to invoke. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
SELECT *
FROM table(pyqTableEval(
   'IRIS',
   '{"modelName":"linregr",
    "datastoreName":"pymodel",
    "oml_connect":1}',
   'XML',
   'myLinearRegressionModel'));
```

The output is the following:

```
NAME  VALUE
----- ------------------------------------------------------------
<root><str>LinearRegression()</str></root>
```

**pyqRowEval Function**

Chunks data into sets of rows and then runs a user-defined Python function on each chunk.

The `pyqRowEval` function passes the data specified by the `INP_QRY` parameter to the Python function specified by the `EXP_NAM` parameter. You can pass arguments to the Python function with the `PAR_QRY` parameter.

The `ROW_NUM` parameter specifies the maximum number of rows to pass to each invocation of the Python function. The last set of rows may have fewer rows than the number specified.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You may define the form of the returned value with the `OUT_QRY` parameter.

**Syntax**

```
pyqRowEval (  
   INP_QRY     VARCHAR2       IN
   PAR_QRY     VARCHAR2       IN
   OUT_QRY     VARCHAR2       IN
)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_QRY</td>
<td>The name of a table or view that specifies the data to pass to the Python function specified by the EXP_NAM parameter. If you're using a table or view owned by another user, use the format <code>&lt;owner name&gt;.&lt;table/view name&gt;</code>. You must have read access to the specified table or view.</td>
</tr>
<tr>
<td>PAR_QRY</td>
<td>A JSON string that contains keyword arguments to pass to the user-defined Python function. Special control arguments, which start with <code>oml_</code>, are not passed to the function specified by EXP_NAM but instead control what happens before or after the invocation of the function.</td>
</tr>
</tbody>
</table>
| OUT_QRY   | One of the following:  
| · A JSON string that specifies the column names and data types of the table returned by pyqRowEval. Any image data is discarded. The Python function must return a `pandas.DataFrame`, a `numpy.ndarray`, a tuple, or a list of tuples.  
| · The name of an existing table or view to use as a prototype. The Python function must return a `pandas.DataFrame`, a `numpy.ndarray`, a tuple, or a list of tuples. If using a table or view owned by another user, use the format `<owner name>.<table/view name>`. You must have read access to the specified table or view.  
| · The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.  
| · The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function in PNG format. |
| ROW_NUM   | The number of rows to include in each invocation of the Python function. |
| EXP_NAM   | The name of a user-defined Python function in the OML4Py script repository. |

Returns

Function `pyqRowEval` returns a table that has the structure specified by the OUT_QRY parameter value.

Example 8-3 Using the pyqRowEval Function

This example loads the Python model `linregr` to predict row chunks of sample iris data. The model is created and saved in the datastore `pymodel` in Example 8-2.

The example defines a Python function and stores it in the OML4Py script repository. It uses the user-defined Python function to create a database table as the result of the `pyqEval` function. It defines a Python function that runs a prediction function on a model loaded from the OML4Py datastore. It then invokes the `pyqTableEval` function to invoke the function on chunks of rows from the database table.
Chapter 8

pyqRowEval Function

In a PL/SQL block, define the function sample_iris_table and store it in the script
repository. The function loads the iris data set, creates two pandas.DataFrame objects,
and then returns a sample of the concatenation of those objects.
BEGIN
sys.pyqScriptCreate('sample_iris_table',
'def sample_iris_table(size):
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
"Sepal_Width","Petal_Length","Petal_Width"])
y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1:
"versicolor",\
2: "virginica"}[x], iris.target)),\
columns = ["Species"])
return pd.concat([y, x], axis=1).sample(int(size))',
FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
Create the SAMPLE_IRIS table in the database as the result of a SELECT statement,
which invokes the pyqEval function on the sample_iris_table user-defined Python
function saved in the script repository with the same name. The sample_iris_table
function returns an iris data sample of size size.
CREATE TABLE sample_iris AS
SELECT *
FROM TABLE(pyqEval(
'{"size":20}',
'{"Species":"varchar2(10)","Sepal_Length":"number",
"Sepal_Width":"number","Petal_Length":"number",
"Petal_Width":"number"}',
'sample_iris_table'));
Define the Python function predict_model and store it with the name linregrPredict
in the script repository. The function predicts the data in dat with the Python model
specified by the modelName argument, which is loaded from the datastore specified by
the datastoreName argument. The predictions are finally concatenated and returned
with dat as the object that the function returns.
BEGIN
sys.pyqScriptCreate('linregrPredict',
'def predict_model(dat, modelName, datastoreName):
import oml
import pandas as pd
objs = oml.ds.load(name=datastoreName, to_globals=False)
pred =
objs[modelName].predict(dat[["Sepal_Length","Sepal_Width",\
"Petal_Length"]])
return pd.concat([dat, pd.DataFrame(pred, \
columns=["Pred_Petal_Width"])], axis=1)',
FALSE, TRUE);

8-10


Run a SELECT statement that invokes the pyqRowEval function, which runs the specified Python function on each chunk of rows in the specified data set.

The INP_QRY argument specifies the data in the SAMPLE_IRIS table to pass to the Python function.

The PAR_QRY argument specifies connecting to the OML4Py server with the special control argument `oml_connect`, passing the input data as a pandas.DataFrame with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

In the OUT_QRY argument, the JSON string specifies the column names and data types of the table returned by pyqRowEval.

The ROW_NUM argument specifies that five rows are included in each invocation of the function specified by EXP_NAM.

The EXP_NAM parameter specifies `linregrPredict`, which is the name in the script repository of the user-defined Python function to invoke.

```
SELECT *
FROM table(pyqRowEval(
    'SAMPLE_IRIS',
    {'oml_connect':1,"oml_input_type":"pandas.DataFrame","modelName":"linregr","datastoreName":"pymodel"},
    {'Species':"varchar2(10)"},
    5,
    'linregrPredict'));
```

The output is the following:

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal_Length</th>
<th>Sepal_Width</th>
<th>Petal_Length</th>
<th>Petal_Width</th>
<th>Pred_Petal_Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>versicolor</td>
<td>5.4</td>
<td>3</td>
<td>4.5</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>versicolor</td>
<td>6</td>
<td>3.4</td>
<td>4.5</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>setosa</td>
<td>5.5</td>
<td>4.2</td>
<td>1.4</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>virginica</td>
<td>6.4</td>
<td>3.1</td>
<td>5.5</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>versicolor</td>
<td>6.1</td>
<td>2.8</td>
<td>4.7</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>setosa</td>
<td>5.4</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>virginica</td>
<td>7.2</td>
<td>3</td>
<td>5.8</td>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>versicolor</td>
<td>6.2</td>
<td>2.2</td>
<td>4.5</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>

The output of `linregrPredict` function for the given data set.
### pyqGroupEval Function

Groups data by one or more columns and runs a user-defined Python function on each group.

The **pyqGroupEval** function runs the user-defined Python function specified by the `EXP_NAM` parameter. You pass data to the Python function with the `INP_QRY` parameter. You can pass arguments to the Python function with the `PAR_QRY` parameter. You specify one or more grouping columns with the `GRP_COL` parameter.

The Python function can return a **boolean**, a **dict**, a **float**, an **int**, a **list**, a **str**, a **tuple** or a **pandas.DataFrame** object. You may define the form of the returned value with the `OUT_QRY` parameter.

**Syntax**

```sql
pyqGroupEval (  
  INP_QRY VARCHAR2 IN,  
  PAR_QRY VARCHAR2 IN,  
  OUT_QRY VARCHAR2 IN,  
  GRP_COL VARCHAR2 IN,  
  EXP_NAM VARCHAR2 IN)  
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_QRY</td>
<td>The name of a table or view that specifies the data to pass to the Python function specified by the EXP_NAM parameter. If using a table or view owned by another user, use the format <code>&lt;owner name&gt;.&lt;table/view name&gt;</code>. You must have read access to the specified table or view.</td>
</tr>
<tr>
<td>PAR_QRY</td>
<td>A JSON string that contains keyword arguments to pass to the user-defined Python function. Special control arguments, which start with <code>oml_</code>, are not passed to the function specified by EXP_NAM, but instead control what happens before or after the invocation of the function. See Also: About Special Control Arguments</td>
</tr>
</tbody>
</table>
| OUT_QRY   | One of the following:  
|           | · A JSON string that specifies the column names and data types of the table returned by `pyqGroupEval`. Any image data is discarded. The Python function must return a `pandas.DataFrame`, a `numpy.ndarray`, a `tuple`, or a list of `tuples`.  
|           | · The name of an existing table or view to use as a prototype. The Python function must return a `pandas.DataFrame`, a `numpy.ndarray`, a `tuple`, or a list of `tuples`. If using a table or view owned by another user, use the format `<owner name>.<table/view name>`. You must have read access to the specified table or view.  
|           | · The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.  
|           | · The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function in PNG format. |
| GRP_COL   | The names of the grouping columns by which to partition the data. Use commas to separate multiple columns. |
| EXP_NAM   | The name of a user-defined Python function in the OML4Py script repository. |

Returns

Function `pyqGroupEval` returns a table that has the structure specified by the OUT_QRY parameter value.

Example 8-4 Using the `pyqGroupEval` Function

This example defines the Python function `create_iris_table` and stores it with the name `create_iris_table` in the OML4Py script repository. It then invokes `pyqEval`, which invokes the user-defined Python function and creates the IRIS database table. The example creates the package `irisPkg` and uses that package in specifying the data cursor to pass to the `irisGroupEval` function, which is a user-defined `pyqGroupEval` function. It defines another Python function, `group_count` and stores it in the script repository with the name `mygroupcount`. The example then invokes the `irisGroupEval` function and passes it the Python function saved with the name `mygroupcount`. 
In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`.

```sql
BEGIN
    sys.pyqScriptCreate('create_iris_table',
        'def create_iris_table():
            from sklearn.datasets import load_iris
            import pandas as pd
            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",
                "Sepal_Width","Petal_Length","Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1:
                "versicolor",
                2: "virginica"}[x], iris.target)),
                        columns = ["Species"])
            return pd.concat([y, x], axis=1)');
END;
/
```

Invoke the `pyqEval` function to create the database table IRIS, using the Python function stored with the name `create_iris_table` in the script repository.

```sql
CREATE TABLE IRIS AS
    (SELECT * FROM pyqEval(
        NULL,
        '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
        "Sepal_Width":"number","Petal_Length":"number",
        "Petal_Width":"number"},
        'create_iris_table'
    ));
```

Define the Python function `group_count` and store it with the name `mygroupcount` in the script repository. The function returns a `pandas.DataFrame` generated on each group of data `dat`.

```sql
BEGIN
    sys.pyqScriptCreate('mygroupcount',
        'def group_count(dat):
            import pandas as pd
            return pd.DataFrame([(dat["Species"][0], dat.shape[0])],
                                columns = ["Species", "CNT"]);'
    END;
/
```

Issue a query that invokes the `pyqGroupEval` function. In the function, the `INP_QRY` argument specifies the data in the IRIS table to pass to the function. The `PAR_QRY` argument specifies the special control argument `oml_input_type`. The `OUT_QRY` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqGroupEval`. The `GRP_COL` parameter specifies the column to group by.
The `EXP_NAM` parameter specifies the user-defined Python function stored with the name `mygroupcount` in the script repository.

```sql
SELECT *
FROM table(
    pyqGroupEval(
        'IRIS',
        '{"oml_input_type":"pandas.DataFrame"}',
        '{"Species":"varchar2(10)", "CNT":"number"}',
        'Species',
        'mygroupcount'));
```

The output is the following.

<table>
<thead>
<tr>
<th>Species</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>50</td>
</tr>
<tr>
<td>versicolor</td>
<td>50</td>
</tr>
<tr>
<td>virginica</td>
<td>50</td>
</tr>
</tbody>
</table>

### sys.pyqScriptCreate Procedure

Creates a user-defined Python function and adds it to the OML4Py script repository.

To create a user-defined Python function, you must have the PYQADMIN database role.

#### Syntax

```sql
sys.pyqScriptCreate (  
    V_NAME          VARCHAR2    IN  
    V_SCRIPT        CLOB        IN  
    V_GLOBAL        BOOLEAN     IN     DEFAULT  
    V_OVERWRITE     BOOLEAN     IN     DEFAULT)
```

#### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_NAME</td>
<td>A name for the user-defined Python function in the OML4Py script repository.</td>
</tr>
<tr>
<td>V_SCRIPT</td>
<td>The definition of the Python function.</td>
</tr>
<tr>
<td>V_GLOBAL</td>
<td>TRUE specifies that the user-defined Python function is public; FALSE specifies that the user-defined Python function is private.</td>
</tr>
<tr>
<td>V_OVERWRITE</td>
<td>If the script repository already has a user-defined Python function with the same name as V_NAME, then TRUE replaces the content of that user-defined Python function with V_SCRIPT and FALSE does not replace it.</td>
</tr>
</tbody>
</table>

#### Example 8-5   Using the pyqScriptCreate Procedure

This example creates a private user-defined Python function named `pyqFun2` in the OML4Py script repository.

```python
BEGIN
    sys.pyqScriptCreate('pyqFun2',
        'def return_frame():
            return
```

---

ORACLE
import numpy as np
import pickle
z = np.array([y for y in zip([str(x) + 'demo' for x in range(10)],
[float(x) / 10 for x in range(10)],
[x for x in range(10)],
[bool(x % 2) for x in range(10)],
[pickle.dumps(x) for x in range(10)],
["test" + str(x**2) for x in range(10)]),
dtype=["a", "U10"], ("b", "f8"), ("c", "i4"), ("d", "?"),
("e", "S20"), ("f", "O")])
return z

This example creates a global user-defined Python function named `pyqFun2` in the script repository and overwrites any existing user-defined Python function of the same name.

BEGIN
sys.pyqScriptCreate('pyqFun2',
'def return_frame():
    import numpy as np
    import pickle
    z = np.array([y for y in zip([str(x) + 'demo' for x in range(10)],
        [float(x) / 10 for x in range(10)],
        [x for x in range(10)],
        [bool(x % 2) for x in range(10)],
        [pickle.dumps(x) for x in range(10)],
        ["test" + str(x**2) for x in range(10)]),
        dtype=["a", "U10"], ("b", "f8"), ("c", "i4"), ("d", "?"),
        ("e", "S20"), ("f", "O"))
    return z',
    TRUE, -- Make the user-defined Python function global. 
    TRUE); -- Overwrite any global user-defined Python function
    -- with the same name.
END;
/

This example creates a private user-defined Python function named `create_iris_table` in the script repository.

BEGIN
sys.pyqScriptCreate('create_iris_table',
'def create_iris_table():
    from sklearn.datasets import load_iris
    import pandas as pd
    iris = load_iris()
    x = pd.DataFrame(iris.data, columns = ["Sepal_Length",
        "Sepal_Width", "Petal_Length", "Petal_Width")
    y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1:
        "versicolor",
        2: 'virginica'}[x], iris.target)),
        columns = ["Species")
    return pd.concat([y, x], axis=1)');
Display the user-defined Python functions owned by the current user.

SELECT * from USER_PYQ_SCRIPTS;

NAME               SCRIPT
----------------- ------------------------------
create_iris_table  def create_iris_table():       from sklearn.datasets
                  import load_iris ...
pyqFun2            def return_frame():       import numpy as np import
                  pickle ... 

Display the user-defined Python functions available to the current user.

SELECT * from ALL_PYQ_SCRIPTS;

OWNER     NAME               SCRIPT
--------   -----------------
OML_USER   create_iris_table  "def create_iris_table():  from
                  sklearn.datasets import load_iris ...
OML_USER   pyqFun2            "def return_frame():  import numpy as np
                  import pickle ...
PYQSYS     pyqFun2            "def return_frame():  import numpy as np
                  import pickle ...

sys.pyqScriptDrop Procedure

Removes a user-defined Python function from the OML4Py script repository.

To drop a user-defined Python function, you must have the PYQADMIN database role.

Syntax

sys.pyqScriptDrop (   V_NAME VARCHAR2 IN   V_GLOBAL BOOLEAN IN DEFAULT   V_SILENT BOOLEAN IN DEFAULT)

Parameter Description
V_NAME A name for the user-defined Python function in the OML4Py script repository.
Parameter Description

V_GLOBAL A BOOLEAN that specifies whether the user-defined Python function to drop is a
global or a private user-defined Python function. The default value is FALSE,
which indicates a private user-defined Python function. TRUE specifies that the
user-defined Python function is public.

V_SILENT A BOOLEAN that specifies whether to display an error message when
sys.pyqScriptDrop encounters an error in dropping the specified user-defined
Python function. The default value is FALSE.

Example 8-6 Using the sys.pyqScriptDrop Procedure

For the creation of the user-defined Python functions dropped in these examples, see
Example 8-5.

This example drops the private user-defined Python function pyqFun2 from the script
repository.

BEGIN
 sys.pyqScriptDrop('pyqFun2');
END;
/

This example drops the global user-defined Python function pyqFun2 from the script
repository.

BEGIN
 sys.pyqScriptDrop('pyqFun2', TRUE);
END;
/

ALL_PYQ_DATASTORE_CONTENTS View

The ALL_PYQ_DATASTORE_CONTENTS view contains information about the contents of
datastores that are available to the current user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null permitted</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNER</td>
<td>VARCHAR2 (128)</td>
<td>NULL</td>
<td>The owner of the datastore.</td>
</tr>
<tr>
<td>DSNAME</td>
<td>VARCHAR2 (128)</td>
<td>NULL</td>
<td>The name of the datastore.</td>
</tr>
<tr>
<td>OBJNAME</td>
<td>VARCHAR2 (128)</td>
<td>NULL</td>
<td>The name of an object in the datastore.</td>
</tr>
<tr>
<td>CLASS</td>
<td>VARCHAR2 (128)</td>
<td>NULL</td>
<td>The class of a Python object in the datastore.</td>
</tr>
<tr>
<td>OBJSIZE</td>
<td>NUMBER</td>
<td>NULL</td>
<td>The size of an object in the datastore.</td>
</tr>
</tbody>
</table>
### COLUMN

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH</td>
<td>NUMBER</td>
<td>NULL permitted</td>
<td>The length of an object in the datastore. The length is 1 for all objects unless the object is a list, dict, pandas.DataFrame, or oml.DataFrame, in which case it is equal to len(obj).</td>
</tr>
<tr>
<td>NROW</td>
<td>NUMBER</td>
<td>NULL permitted</td>
<td>The number of rows of an object in the datastore. The number is 1 for all objects except for pandas.DataFrame and oml.DataFrame objects, in which case it is equal to len(df).</td>
</tr>
<tr>
<td>NCOL</td>
<td>NUMBER</td>
<td>NULL permitted</td>
<td>The number of columns of an object in the datastore. The number is len(obj) if the object is a list or dict, len(obj.columns) if the object is a pandas.DataFrame or oml.DataFrame, and 1 otherwise.</td>
</tr>
</tbody>
</table>

### Example 8-7  Selecting from the ALL_PYQ_DATASTORE_CONTENTS View

This example selects all columns from the ALL_PYQ_DATASTORE_CONTENTS view. For the creation of the datastores in this example, see Example 3-14.

```
SELECT * FROM ALL_PYQ_DATASTORE_CONTENTS
```

<table>
<thead>
<tr>
<th>DSOWNER</th>
<th>DSNAME</th>
<th>OBJNAME</th>
<th>CLASS</th>
<th>OBJSIZE</th>
<th>LENGTH</th>
<th>NROW</th>
<th>NCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>OML_USER</td>
<td>ds_pydata</td>
<td>oml_boston</td>
<td>oml.DataFrame</td>
<td>1073</td>
<td>506</td>
<td>506</td>
<td>14</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_pydata</td>
<td>oml_diabetes</td>
<td>oml.DataFrame</td>
<td>964</td>
<td>442</td>
<td>442</td>
<td>11</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_pydata</td>
<td>wine</td>
<td>Bunch</td>
<td>24177</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_pymodel</td>
<td>regr1</td>
<td>LinearRegression</td>
<td>706</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_pymodel</td>
<td>regr2</td>
<td>oml.glm</td>
<td>5664</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_wine_data</td>
<td>oml_wine</td>
<td>oml.DataFrame</td>
<td>1410</td>
<td>178</td>
<td>178</td>
<td>14</td>
</tr>
</tbody>
</table>

### ALL_PYQ_DATASTORES View

The ALL_PYQ_DATASTORES view contains information about the datastores that are available to the current user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSOWNER</td>
<td>VARCHAR2(256)</td>
<td>NULL permitted</td>
<td>The owner of the datastore.</td>
</tr>
</tbody>
</table>
### Table: All_PyQ_Datastores View Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNAME</td>
<td>VARCHAR2(128)</td>
<td>NULL</td>
<td>The name of the datastore.</td>
</tr>
<tr>
<td>NOBJ</td>
<td>NUMBER</td>
<td>NULL</td>
<td>The number of objects in the datastore.</td>
</tr>
<tr>
<td>DSSIZE</td>
<td>NUMBER</td>
<td>NULL</td>
<td>The size of the datastore.</td>
</tr>
<tr>
<td>CDATE</td>
<td>DATE</td>
<td>NULL</td>
<td>The date on which the datastore was created.</td>
</tr>
<tr>
<td>GRANTABLE</td>
<td>VARCHAR2(1)</td>
<td>NULL</td>
<td>Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.</td>
</tr>
</tbody>
</table>

### Example 8-8 Selecting from the ALL_PYQ_DATASTORES View

This example selects all columns from the ALL_PYQ_DATASTORES view. It then selects only the DSNAME and GRANTABLE columns from the view. For the creation of the datastores in these examples, see Example 3-14.

```sql
SELECT * FROM ALL_PYQ_DATASTORES;
```

<table>
<thead>
<tr>
<th>DSOWNER</th>
<th>DSNAME</th>
<th>NOBJ</th>
<th>DSSIZE</th>
<th>CDATE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OML_USER</td>
<td>ds_pydata</td>
<td>3</td>
<td>26214</td>
<td>18-MAY-19</td>
<td>python</td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_pymodel</td>
<td>2</td>
<td>6370</td>
<td>18-MAY-19</td>
<td></td>
</tr>
<tr>
<td>OML_USER</td>
<td>ds_wine_data</td>
<td>1</td>
<td>1410</td>
<td>18-MAY-19</td>
<td>wine</td>
</tr>
</tbody>
</table>

This example selects only the DSNAME and GRANTABLE columns from the view.

```sql
SELECT DSNAME, GRANTABLE FROM ALL_PYQ_DATASTORES;
```

<table>
<thead>
<tr>
<th>DSNAME</th>
<th>GRANTABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_pydata</td>
<td>F</td>
</tr>
<tr>
<td>ds_pymodel</td>
<td>T</td>
</tr>
<tr>
<td>ds_wine_data</td>
<td>F</td>
</tr>
</tbody>
</table>
ALL_PYQ_SCRIPTS View

The ALL_PYQ_SCRIPTS view contains information about the user-defined Python functions in the OML4Py script repository that are available to the current user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
<td>VARCHAR2(256)</td>
<td>NULL</td>
<td>The owner of the user-defined Python function.</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2(128)</td>
<td>NULL</td>
<td>The name of the user-defined Python function.</td>
</tr>
<tr>
<td>SCRIPT</td>
<td>CLOB</td>
<td>NULL</td>
<td>The user-defined Python function.</td>
</tr>
</tbody>
</table>

Example 8-9  Selecting from the ALL_PYQ_SCRIPTS View

This example selects the owner and the name of the user-defined Python function from the ALL_PYQ_SCRIPTS view. For the creation of the user-defined Python functions, see Example 8-5.

```sql
SELECT owner, name FROM ALL_PYQ_SCRIPTS;
```

```
OWNER      NAME
--------   -----------------
OML_USER   create_iris_table
OML_USER   tmpqfun2
PYQSYS     tmpqfun2
```

This example selects the name of the user-defined Python function and the function definition from the view.

```sql
SELECT name, script FROM ALL_PYQ_SCRIPTS WHERE name = 'create_iris_table';
```

```
NAME          SCRIPT
--------------
create_iris_table  "def create_iris_table(): from sklearn.datasets import load_iris ..."
USER_PYQ_DATASTORES View

The USER_PYQ_DATASTORES view contains information about the datastores that are owned by the current user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNAME</td>
<td>VARCHAR2(128)</td>
<td>NULL</td>
<td>The name of the datastore.</td>
</tr>
<tr>
<td>NOBJ</td>
<td>NUMBER</td>
<td>NULL</td>
<td>The number of objects in the datastore.</td>
</tr>
<tr>
<td>DSSIZE</td>
<td>NUMBER</td>
<td>NULL</td>
<td>The size of the datastore.</td>
</tr>
<tr>
<td>CDATE</td>
<td>DATE</td>
<td>NULL</td>
<td>The date on which the datastore was created.</td>
</tr>
<tr>
<td>GRANTABLE</td>
<td>VARCHAR2(1)</td>
<td>NULL</td>
<td>Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.</td>
</tr>
</tbody>
</table>

Example 8-10 Selecting from the USER_PYQ_DATASTORES View

This example selects all columns from the USER_PYQ_DATASTORES view. For the creation of the datastores in these examples, see Example 3-14.

```
SELECT * FROM USER_PYQ_DATASTORES;
```

<table>
<thead>
<tr>
<th>DSNAME</th>
<th>NOBJ</th>
<th>DSSIZE</th>
<th>CDATE</th>
<th>DESCRIPTION</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_wine_data</td>
<td>1</td>
<td>1410</td>
<td>18-MAY-19</td>
<td>wine dataset</td>
<td>F</td>
</tr>
<tr>
<td>ds_pydata</td>
<td>3</td>
<td>26214</td>
<td>18-MAY-19</td>
<td>python datasets</td>
<td>F</td>
</tr>
<tr>
<td>ds_pymodel</td>
<td>2</td>
<td>6370</td>
<td>18-MAY-19</td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

This example selects only the DSNAME and GRANTABLE columns from the view.

```
SELECT DSNAME, GRANTABLE FROM USER_PYQ_DATASTORES;
```

<table>
<thead>
<tr>
<th>DSNAME</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_wine_data</td>
<td>F</td>
</tr>
<tr>
<td>ds_pydata</td>
<td>F</td>
</tr>
<tr>
<td>ds_pymodel</td>
<td>T</td>
</tr>
</tbody>
</table>
USER_PYQ_SCRIPTS View

This view contains information about the user-defined Python functions in the OML4Py script repository that are owned by the current user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Null</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>VARCHAR2(128)</td>
<td>NOT NULL</td>
<td>The name of the user-defined Python function.</td>
</tr>
<tr>
<td>SCRIPT</td>
<td>CLOB</td>
<td>NULL</td>
<td>permitted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The user-defined Python function.</td>
</tr>
</tbody>
</table>

Example 8-11 Selecting from the USER_PYQ_SCRIPTS View

This example selects all columns from USER_PYQ_SCRIPTS. For the creation of the user-defined Python functions, see Example 8-5.

```sql
SELECT * FROM USER_PYQ_SCRIPTS;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>SCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_iris_table</td>
<td>&quot;def create_iris_table(): from sklearn.datasets import load_iris ...</td>
</tr>
<tr>
<td>tmpqfun2</td>
<td>&quot;def return_frame(): import numpy as np import pickle ...</td>
</tr>
</tbody>
</table>
Index

A

algorithm selection class, 6-6
algorithms
Apriori, 5-19
attribute importance, 5-16
Automated Machine Learning, 6-1
automatically selecting, 6-16
Decision Tree, 5-25
Expectation Maximization, 5-32
Explicit Semantic Analysis, 5-47
Generalized Linear Model, 5-53
k-Means, 5-63
machine learning, 5-1
Minimum Description Length, 5-16
Naive Bayes, 5-69
Neural Network, 5-77
Random Forest, 5-87
settings common to all, 5-4
Singular Value Decomposition, 5-95
Support Vector Machine, 5-101
ALL_PYQ_DATASTORE_CONTENTS view, 8-18
ALL_PYQ_DATASTORES view, 8-19
ALL_PYQ_SCRIPTS view, 8-21
anomaly detection models, 5-101
Apriori algorithm, 5-19
attribute importance, 5-16
Automated Machine Learning
about, 6-1
Automatic Machine Learning
collection parameter, 3-1
Autonomous Database, 3-1

D

data
about moving, 3-8
exploring, 4-17
filtering, 4-13
preparing, 4-1
selecting, 4-3
data parallel processing, 7-1
database
collection to an on-premises, 3-4
datastores
about, 3-20
database views for, 8-18, 8-19, 8-22
deleting objects, 3-28
describing objects in, 3-27
getting information about, 3-25

classes (continued)
oml.dt, 5-25
oml.em, 5-32
oml.em, 5-32
oml.glm, 5-53
oml.graphics, 4-31
oml.km, 5-63
oml.nb, 5-69
oml.nn, 5-77
oml.rf, 5-87
oml.svd, 5-95
oml.svm, 5-101
classification algorithm, 5-87
classification models, 5-25, 5-53, 5-69, 5-77, 5-87, 5-101
client
installing for Linux on-premises, 2-10
classification models, 5-32, 5-47, 5-63
connection
collection a on-premises database, 3-4
functions, 3-1
control arguments, 7-1
cx_Oracle package, 3-1
cx_Oracle.connect function, 3-1

C

classes
Automated Machine Learning, 6-1
GlobalFeatureImportance, 5-10
machine learning, 5-1
oml.ai, 5-16
oml.ar, 5-19
oml.automl.AlgorithmSelection, 6-6
oml.automl.FeatureSelection, 6-8
oml.automl.ModelSelection, 6-16
oml.automl.ModelTuning, 6-11
datastores (continued)
  granting or revoking access to, 3-30
  loading objects from, 3-24
  saving objects in, 3-21
Decision Tree algorithm, 5-25
  dropping
    tables, 3-15

E

EM model, 5-32
Embedded Python Execution
  about, 7-1
  about the SQL interface for, 8-1
  SQL interface for, 8-1
ESA model, 5-47
Expectation Maximization algorithm, 5-32
explainability, 5-10
Explicit Semantic Analysis algorithm, 5-47
exporting models, 5-6

F

  feature extraction algorithm, 5-47
  feature extraction class, 5-95
  feature selection class, 6-8
  functions
    cx_Oracle.connect, 3-1
    Embedded Python Execution, 7-1
    for graphics, 4-31
    for managing user-defined Python functions, 7-18
    oml.boxplot, 4-31
    oml.check_embed, 3-1, 3-4
    oml.connect, 3-1, 3-4
    oml.create, 3-15
    oml.cursor, 3-8, 3-15
    oml.dir, 3-8, 3-13
    oml.disconnect, 3-1, 3-4
    oml.do_eval, 7-3
    oml.drop, 3-15
    oml.ds.delete, 3-28
    oml.ds.describe, 3-27
    oml.ds.dir, 3-25
    oml.ds.load, 3-24
    oml.ds.save, 3-21
    oml.grant, 3-30
    oml.group_apply, 7-8
    oml.hist, 4-31
    oml.index_apply, 7-16
    oml.isconnected, 3-1, 3-4
    oml.row_apply, 7-12
    oml.script.create, 7-18
    oml.script.dir, 7-23
    oml.script.drop, 7-25

functions (continued)
  oml.script.load, 7-24
  oml.set_connection, 3-1
  oml.sync, 3-13
  oml.table_apply, 7-5
  pyqEval, 8-2
  pyqGroupEval, 8-12
  pyqRowEval, 8-8
  pyqTableEval, 8-5

G

GLM models, 5-53
granting
  access to scripts and datastores, 3-30
  user privileges, 2-6
graphics
  rendering, 4-31

I

importing models, 5-6
installing
  client for Linux on-premises, 2-10
  server for Linux on-premises, 2-4
Instant Client
  installing for Linux on-premises, 2-9

K

KM model, 5-63

L

libraries inOML4Py, 1-8
Linux
  installing Python for, 2-2
  requirements, 2-1
  uninstalling on-premises client for, 2-13
  uninstalling on-premises server for, 2-8
Linux on-premises
  installing client for, 2-10
  installing Oracle Instant Client for, 2-9
  installing server for, 2-4
  supporting packages for, 2-3

M

machine learning
  classes, 5-1
methods
  drop, 4-13
  drop_duplicates, 4-13
  dropna, 4-13
methods (continued)

for exploring data, 4-17
for preparing data, 4-1
pull, 3-11
Minimum Description Length algorithm, 5-16
model selection, 6-16
model tuning, 6-11
models

association rules, 5-19
attribute importance, 5-16
Decision Tree, 5-25
Expectation Maximization, 5-32
explainability, 5-10
Explicit Semantic Analysis, 5-47
exporting and importing, 5-6
for anomaly detection, 5-101
for classification, 5-25, 5-53, 5-57, 5-69, 5-77, 5-87, 5-101
for clustering, 5-32, 5-63
for feature extraction, 5-47, 5-95
for regression, 5-53, 5-77, 5-101
Generalized Linear Model, 5-53
k-Means, 5-63
Naive Bayes, 5-69
Neural Network, 5-77
parametric, 5-53
persisting, 5-1
Random Forest, 5-87
Singular Value Decomposition, 5-95
Support Vector Machine, 5-101

moving data

about, 3-8
to a local Python session, 3-11
to the database, 3-9

N

Naive Bayes model, 5-69
Neural Network model, 5-77

O

oml_input_type argument, 7-1
oml_na_omit argument, 7-1
oml.ai class, 5-16
oml.ar class, 5-19
oml.automl.AlgorithmSelection class, 6-6
oml.automl.FeatureSelection class, 6-8
oml.automl.ModelSelection class, 6-16
oml.automl.ModelTuning class, 6-11
oml.boxplot function, 4-31
oml.check_embed function, 3-1, 3-4
oml.connect function, 3-1, 3-4
oml.create function, 3-15
oml.cursor function, 3-8, 3-15
oml.dir function, 3-8, 3-13
oml.disconnect function, 3-1, 3-4
oml.do_eval function, 7-3
oml.drop function, 3-15
oml.ds.delete function, 3-28
oml.ds.describe function, 3-27
oml.ds.dir function, 3-25
oml.ds.load function, 3-24
oml.ds.save function, 3-21
oml.dt class, 5-25
oml.em class, 5-32
oml.esa class, 5-47
oml.glm class, 5-53
oml.grant function, 3-30
oml.graphics class, 4-31
oml.group_apply function, 7-8
oml.hist function, 4-31
oml.index_apply function, 7-16
oml.isconnected function, 3-1, 3-4
oml.km class, 5-63
oml.nb class, 5-69
oml.nn class, 5-77
oml.push function, 3-9
oml.revoke function, 3-30
oml.rf class, 5-87
oml.row_apply function, 7-12
oml.script.create function, 7-18
oml.script.dir function, 7-23
oml.script.drop function, 7-25
oml.script.load function, 7-24
oml.set_connection function, 3-1, 3-4
oml.svd class, 5-95
oml.svm class, 5-101
oml.sync function, 3-13
oml.table_apply function, 7-5
OML4Py, 1-1
on-premises client
installing, 2-9
uninstalling, 2-13
on-premises server
installing, 2-4
uninstalling, 2-8
on-premises system requirements, 2-1
Oracle Machine Learning Notebooks, 3-1
Oracle Machine Learning Python interpreter, 3-1
Oracle wallets
about, 3-3

P

packages

supporting for Linux on-premises, 2-3
parallel processing, 7-1
parametric models, 5-53
PL/SQL procedures
  sys.pyqScriptCreate, 8-15
  sys.pyqScriptDrop, 8-17
predict method, 5-69
predict.proba method, 5-69
privileges
  required, 2-6
proxy objects, 1-3
  for database tables, 3-13, 3-15
  storing, 3-20
pull method, 3-11
PYQADMIN role, 2-6
pyqEval function, 8-2
pyqGroupEval function, 8-12
pyqRowEval function, 8-8
pyqTableEval function, 8-5
pyquser.sql script, 2-7
Python
  installing for Linux, 2-2
  libraries in OML4Py, 1-8
  requirements, 2-1
  version used, 1-8
Python interpreter, 3-1
Python objects
  storing, 3-20
Python to SQL conversion, 1-3
R
  Random Forest algorithm, 5-87
  ranking
    attribute importance, 5-16
  read privilege
    granting or revoking, 3-30
  regression models, 5-53, 5-77
  requirements
    on-premises system, 2-1
  resources
    managing, 1-5
  revoking
    access to scripts and datastores, 3-30
roles
  PYQADMIN, 2-6
S
  scoring new data, 1-2, 5-1
  script repository
    granting or revoking access to, 3-30
    managing user-defined Python functions in, 7-18
    registering a user-defined function, 7-18
scripts
  pyquser, 2-7
server
  installing for Linux on-premises, 2-4
settings
  about model, 5-3
  Apriori algorithm, 5-19
  association rules, 5-19
  Decision Tree algorithm, 5-25
  Expectation Maximization model, 5-32
  Explicit Semantic Analysis algorithm, 5-47
  Generalized Linear Model algorithm, 5-53
  k-Means algorithm, 5-63
  Minimum Description Length algorithm, 5-16
  Naive Bayes algorithm, 5-69
  Neural Network algorithm, 5-77
  Random Forest algorithm, 5-87
  shared algorithm, 5-4
  Singular Value Decomposition algorithm, 5-95
  attribute importance, 5-16
  Support Vector Machine algorithm, 5-101
  special control arguments, 7-1
SQL APIs
  pyqEval function, 8-2
  pyqGroupEval function, 8-12
  pyqRowEval function, 8-8
  pyqTableEval function, 8-5
SQL to Python conversion, 1-3
supporting packages
  for Linux on-premises, 2-3
  SVD model, 5-95
  SVM models, 5-101
  synchronizing database tables, 3-13
  sys.pyqScriptCreate procedure, 8-15
  sys.pyqScriptDrop procedure, 8-17
T
  tables
    creating, 3-15
    dropping, 3-13, 3-15
    proxy objects for, 3-13, 3-15
  task parallel processing, 7-1
  transparency layer, 1-3
U
  uninstalling
    on-premises client, 2-13
    on-premises server, 2-8
  USER_PYQ_DATASTORES view, 8-22
  USER_PYQ_SCRIPTS view, 8-23
  user-defined Python functions
    Embedded Python Execution of, 7-1
  users
    creating new, 2-7
views

ALL_PYQ_DATASTORE_CONTENTS, 8-18
ALL_PYQ_DATASTORES, 8-19
ALL_PYQ_SCRIPTS, 8-21
USER_PYQ_DATASTORES, 8-22

views (continued)
USER_PYQ_SCRIPTS, 8-23

W

wallets

about Oracle, 3-3