MySQL AI User Guide

Abstract

This document describes how to use MySQL AI. It covers how to load data, run queries, optimize analytics workloads, and use machine learning and generative AI capabilities.

For legal information, see the Preface and Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2025-09-15 (revision: 83520)

Table of Contents

Preface and Legal Notices

This is the user manual for MySQL AI.

Licensing information. This product may include third-party software, used under license. See the MySQL AI License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this MySQL AI release.

Legal Notices

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 for Accessibility

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.

Chapter 1 Introduction to MySQL AI

This chapter describes MySQL AI.

MySQL AI consists of the following components:

- MySQL Enterprise Edition for MySQL AI, which contains the following AI components:
  - MySQL Commercial Server
  - AI Engine
  - MySQL AI Plugin
- MySQL Shell AI Edition, which supports MySQL Shell Workbench.
- MySQL Router AI Edition, which supports the MySQL REST Service

Important

The responses produced by generative artificial intelligence (AI) models may not always be accurate, complete, current, or appropriate for Your intended use. You are responsible for Your use of AI output and for reviewing and independently verifying the accuracy and appropriateness of AI output before Your use. AI output may not be unique, and other customers may receive similar output.

Chapter 2 Installing MySQL AI

Table of Contents

This chapter describes how to install MySQL AI.

2.1 Supported Platforms and Requirements

MySQL AI is supported only on the following:

- Oracle Linux 8
- Red Hat Enterprise Linux 8

The installation requires the following:

- Your system satisfies the platform requirements and the following system requirements for MySQL AI:
  - CPUs: 32 logical, or virtual, CPU cores
  - RAM: 128GB
  - Storage: 512GB
- You have a license for the MySQL Enterprise Edition.
- You are a sudoer on your system.
- You have TLS certificates and keys that satisfy the MySQL requirements for them, if you want to configure encrypted communication with the MySQL AI components using your own certificates and keys.

> **Warning**
>
> In this installation of MySQL AI, the MySQL Shell GUI server and the MySQL REST server both run on the same host as the MySQL Server, and they allow a user to access the MySQL server through them from a remote host, even if the user has been restricted to connect only from `localhost` (or 127.0.0.1 for IPv4, or ::1 for IPV6). System administrators may want to prevent that from happening, especially if their systems are going on production. Possible measures that can be taken include:
>
> - Disallow certain users (for example, the server administrator) from connecting by HTTP connections, but only allow connections by, for example, Unix sockets with the `auth_socket` authentication plugin. See Socket Peer-Credential Pluggable Authentication.
> - Do not install the MySQL Shell GUI and MySQL REST Service.

2.2 Installing MySQL AI

This section describes how to install MySQL AI:

To install the MySQL AI, installer do the following:

1. Download the MySQL AI RPM bundles from My Oracle Support (MOS) or Oracle Software Delivery Cloud.

2. Extract all RPMs from the bundle:

   ```
   $> tar -xvf mysql-ai-version.distro.arch.rpm-bundle.tar
   ```

3. Install the MySQL AI Installer with the following command:

   ```
   $> sudo dnf localinstall mysql-ai-setup-version.distro.arch.rpm
   ```

   The MySQL AI Installer is installed.

4. Run the MySQL AI Installer to install and configure all the components of MySQL AI. It can be run in GUI mode or in command-line mode.

2.2.1 MySQL AI GUI Installation

To run the MySQL AI installer GUI, start the installer GUI in the folder where you have extracted the RPMs with the following command:

```
$> sudo mysql-ai-setup
```

Note

You must run the installer in the same directory as the extracted RPMs.

The installer guides you through the following configuration pages:

1. **Introduction**: Click **Continue**.

2. **System Requirements**: Checks your system for hardware requirements. Click **Continue** to proceed if your system meets all the minimum requirements.

   A report is given if the minimum requirements are not satisfied. In that case, choose **Continue Anyway** or **Cancel**.

   Warning

   MySQL AI might not work or experience performance issues if installed on a system that does not satisfy the minimum requirements.

   The installer also checks if any default ports used for communication with the MySQL AI components are already in use, and reports to you if that is the case.

3. **User & password**: Define a user name and password for the MySQL root user. The password must satisfy the MEDIUM level policy of the validate_password component.

   You can choose to **Only allow local connections for this user** (see the Warning near the beginning of Chapter 2, *Installing MySQL AI*).

Click **Continue** to proceed.

4. **MySQL Studio**: Install and configure the MySQL Studio.

   You can replace the default port number (8080). A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.

5. **Router & Shell**: Install and configure the MySQL Shell GUI and MySQL Router for MySQL AI

   > **Note**
   >
   > Check Warning before proceeding with the installation of the MySQL Shell GUI and MySQL Router (MySQL REST Service).

   Select to install both, either, or neither, by going through the following pages:

   - **MySQL Shell GUI**: Select **Install the MySQL Shell GUI web service** to install the component.

     You can replace the default port number (8000) with another number for MySQL Shell GUI web server to listen for connections. A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.

   - **MySQL Router (MySQL REST Service)**: Select **Install MySQL Router and configure it for MySQL REST Service** to install the component.

     You can replace the default HTTPS port number (8443) with another number for the MySQL REST Service web server to listen to connections. A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.

     You can enter a secret for JSON Web Secret (JWS) tokens. If you do not enter one, a random secret will be created.

   Click **Continue** to proceed.

6. **Vector Store**: Specify the directory for loading documents into the vector store. The location must be configured by the server system variable `secure_file_priv` for `mysqld` to import data securely from it. The default location is `/var/lib/mysql-files`. If you specify a directory that does not exist, it will be created.

   Click **Continue** to proceed.

7. **Certificates**: Configure TLS certificates for encrypted communication with each of the following components of MySQL AI.

   > **Note**
   >
   > - The certificates and keys you provide must satisfy MySQL requirements. See Creating SSL and RSA Certificates and Keys.
   > - The certificate, key, and bundle files specified must be readable by the `root` user who installs MySQL AI; adjust their file permissions if needed.
   > - The certificate, key, and bundle files must not be passphrase protected.
   > - A file path to a certificate bundle file is expected in the certificate field. However, the path can also point to either a certificate file or a bundle file that does not contain the private key, in which case a separate field appears for

you to provide the file path for the private key or, for the PEM format only, the actual key string (pasted keys are represented by icons on the screen).

- The Common Name (CN) for your certificate is shown. The user can verify that the CN is correct and, for the MySQL AI plugin and MySQL Machine Learning Services, correct it if the installer misreads it.

- **MySQL Server**: Provide the path to the certificate bundle in PEM or PKSC#12 format for communication between the server and other components using the `mysql` and `mysqlx` protocols. If no certificate is supplied, a self-signed certificate is generated.

- **MySQL Server Plugin** (for MySQL AI) and **MySQL Machine Learning Services**: Provide paths to the certificate bundles in PEM or PKSC#12 format. Two distinct certificate bundles are required for the two components. If no certificates are provided, encrypted communication between MySQL AI components will be disabled.

- **MySQL Studio**,**MySQL Shell GUI** and **MySQL REST Service**: Provide the paths to the certificate bundles in PEM or PKSC#12 format. If either of the certificates is not supplied, a self-signed certificate will be created for the respective service.

Click **Continue** to proceed.

8. **Finalize Installation**: Confirm selections and begin the installation procedure. The following issues are reported if they occur:

   - **Networking ports are assigned multiple times**. Use the **Previous** button to go back to earlier pages and correct the port assignments.

   - **Internal communication between MySQL Server and the Machine Learning and AI subsystem should not be encrypted** because no certificates were given. Use the **Previous** button to go back and supply the certificates, or select the note to confirm this.

   Click **Finalize** to start installation of MySQL AI.

Finalizing Installation

The installer completes and presents a message containing information on URLs and endpoints for the selected components.

For example, if you selected MySQL Studio, MySQL Shell Workbench, and MySQL Router (MySQL REST Service):

```
==============================================================================
Installation finished.
==============================================================================

To access MySQL Studio, navigate to the following
URL in a web browser:

    https://hostAddress:8080/

To access a SQL shell for this MySQL AI instance, navigate to the following
URL in a web browser:

    https://hostAddress:8000/

The MySQL REST Service endpoint is:

    https://hostAddress:8443/
```

```
================================================================================
```

Customizing the GUI Installer

The installer GUI can also take command-line installer options to populate fields, skip specific elements of the installation, and so on. The following example instructs the installer to run without the option to install MySQL Studio and MySQL Router, and sets the root username to John:

```
        sudo mysql-ai-setup --skip-mysql-studio --skip-mysql-router --mysql-root-user=John
```

2.2.2 Command-line Installation

The MySQL AI Installer can also be run in command-line mode, without invoking the installation GUI. Execute the following command in the folder where you have extracted the RPMs from the MySQL AI RPM bundle:

```
$> sudo mysql-ai-setup --cli [options]
options:
        --option-long-name[=value-list]
    |   -option-short-name [value-list]

value-list:
    value[,value[,...]]
```

The command options are described in groups below (use the `-h` or `--help` option to see the option descriptions):

Installation Type

- `--skip-install`: Do not install anything. This is useful for testing system requirements and installation options.

Install Without Satisfying Minimum Requirements

- `--skip-requirements`: Install even if the system does not satisfy the minimum requirements.

Warning

MySQL AI might not work or might have performance issues if installed on a system that does not satisfy the minimum requirements.

User and Password

- `--mysql-root-user=`*`username`*: User name and password for the MySQL root user.
- `--mysql-root-password=`*`password`*: Password for the MySQL root user. The password must satisfy the MEDIUM level policy of the validate_password component.
- `--mysql-root-allow-remote-connection`: The root user is allowed to connect from hosts other than `localhost`. See the Warning near the beginning of Chapter 2, *Installing MySQL AI*.

MySQL Studio, MySQL Shell Workbench and MySQL Router (MySQL REST Service)

Note

Check the Warning near the beginning of Chapter 2, *Installing MySQL AI* before installing the MySQL Shell GUI and MySQL Router (MySQL REST Service).

- `--install-mysql-studio`: Install the MySQL Studio service.
- `--mysql-studio-port=`*`port#`*: Replace the default port number (8000) with another one for MySQL Studio's server to listen for connections. A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.
- `--skip-mysql-studio`: Skip installing MySQL Studio.
- `--install-mysql-shell-gui`: Install the MySQL Shell Workbench service.
- `--skip-mysql-shell-gui`: Skip installing MySQL Shell Workbench.
- `--mysql-shell-gui-port=`*`port#`*: Replace the default port number (8000) with another one for MySQL Shell GUI web server to listen for connections. A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.
- `--skip-mysql-router`: Skip installing MySQL Router and MySQL REST Service.
- `--mysql-router-port=`*`port#`*: Replace the default HTTPS port number (8443) with another one for the MySQL REST Service web server to listen to connections. A warning is displayed if the port you entered is already in use or will be used by another MySQL AI component.
- `--mysql-router-jwt-secret=`*`jwt-secret`*: Provide a secret for JSON Web Secret (JWS) tokens. If this option is not specified, a random secret will be created by default.

Vector Store

- `--secure-file-priv=`*`filepath`*: Specify the directory for loading documents into the vector store. The location must be configured by the server system variable `secure_file_priv` for `mysqld` to import data securely from it. If the option is not specified, the default location is `/var/lib/mysql-files`. If you specify a directory that does not exist, it will be created.

Certificates

Configure TLS certificates for encrypted communication with each of the following components of MySQL AI.

Notes

- The certificate, key, and bundle files specified must be readable by `root` user who installs MySQL AI; adjust their file permissions if needed.
- The certificate, key, and bundle files must **not** be passphrase protected.
- A file path to a certificate bundle file is expected in the `*-certificate` option. However, the path can also point to either a certificate file or a bundle file that does not contain the private key, in which case use the `*-private-key` to provide the file path for the private key or, for the PEM format only, the actual key string.

Certificates for MySQL Server. Provide the certificate and private key in PEM or PKSC#12 format for communication with MySQL Server using the `mysql` and `mysqlx` protocols. If no certificate is supplied, a self-signed certificate is generated.

- `--mysql-server-tls-certificate=`*`filepath`*: Location of the certificate bundle used for HTTPS communication by MySQL Server.

- `--mysql-server-tls-private-key=`*`filepath`*: The private key used for HTTPS communication by MySQL Server. This option is needed only if `--mysql-server-tls-certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for PEM format only, the actual key string.

Certificates for AI Plugin and Machine Learning Services. Provide the certificates in PEM or PKSC#12 format. **Two distinct certificate bundles are required for the two components**. If no certificates and keys are provided for any of the two components, encrypted communication with the component is disabled, unless self-signed certificates, with specified common names, are requested.

- `--skip-ai-encryption`: Use this option to explicitly turn off encryption for communication with the AI plugin and Machine Learning services. If this command line option is absent, installer will quit without installing MySQL AI unless certificates are provided or self-signed certificates are requested (see options below).
- `--ai-plugin-certificate=`*`filepath`*: Location of the certificate bundle used for HTTPS communication with the AI plugin.
- `--ai-plugin-private-key=`*`filepath`*: The private key used for HTTPS communication with the AI plugin. This option is needed only if `--ai-plugin-certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for PEM format only, the actual key string.
- `--ai-plugin-common-name=`*`string`*: Common name for the certificate for communication with the AI plugin. This option is only needed if you want to correct the installer's reading of the common name from your certificate.
- `--ai-plugin-create-self-signed-certificate=`*`Common_Name`*: Create a self-signed certificate for communication with the AI plugin with the common name specified by this option.
- `--ai-services-certificate=`*`filepath`*: Location of the certificate bundle used for HTTPS communication with the Machine Learning Service.
- `--ai-services-private-key=`*`filepath`*. The private key used for HTTPS communication with the AI plugin. This option is needed only if `--ai-services-certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for the PEM format only, the actual key string.
- `--ai-services-common-name=`*`string`*: Common name for the certificate for communication with the Machine Learning service. This option is only needed if you want to correct the installer's reading of the common name from your certificate.
- `--ai-services-create-self-signed-certificate=`*`Common_Name`*: Create a self-signed certificate for communication with the Machine Learning service with the common name specified by this option.

Certificates for MySQL Studio, MySQL Shell Workbench, and MySQL Router (MySQL REST Service): Provide the certificate and private key in PEM or PKSC#12 format. If either of the certificates is not supplied, a self-signed certificate will be created for the respective service.

- `--mysql-studio-https-certificate=`*`filepath`*: Location of the certificate bundle used for HTTPS communication by the MySQL Studio.
- `--mysql-studio-https-private-key=`*`filepath`*: The private key used for HTTPS communication by MySQL Studio. This option is needed only if `--mysql-studio-https-`

`certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for the PEM format only, the actual key string.

- `--mysql-shell-https-certificate`=*`filepath`*: Location of the certificate bundle used for HTTPS communication by the MySQL Shell Workbench service.
- `--mysql-shell-https-private-key`=*`filepath`*: The private key used for HTTPS communication by the MySQL Shell Workbench service. This option is needed only if `--mysql-shell-https-certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for the PEM format only, the actual key string.
- `--mysql-router-https-certificate`=*`filepath`*: Location of the certificate bundle used for HTTPS communication by MySQL Router (MySQL REST Service).
- `--mysql-router-https-private-key`=*`filepath`*: The private key used for HTTPS communication by MySQL Router (MySQL REST Service). This option is needed only if `--mysql-router-https-certificate` points to a certificate file, or a bundle file that does not contain the private key. Provide with this option the file path for the private key or, for the PEM format only, the actual key string.

Chapter 3 Loading Data in MySQL AI

Table of Contents

The sections in this chapter describe how to load data in MySQL AI.

3.1 Bulk Ingest Data

MySQL includes a bulk load extension to the `LOAD DATA` statement. It can do the following:

- Optimize the loading of data sorted by primary key.
- Optimize the loading of unsorted data.
- Optimize the loading of data from an object store.
- Optimize the loading of data from a series of files.
- Load a MySQL Shell dump file.
- Load ZSTD compressed CSV files.
- Monitor bulk load progress with the Performance Schema.
- Large data support.

Use a second session to monitor bulk load progress:

- If the data is sorted, there is a single stage: `loading`.
- If the data is unsorted, there are two stages: `sorting` and `loading`.

Bulk Ingest Data Type Support

`LOAD DATA` with `ALGORITHM=BULK` supports tables with at least one column with the `VECTOR` data type. If you attempt to load a table without at least one column with the `VECTOR` data type, an error occurs.

In addition to the requirement to have at least one `VECTOR` column, `LOAD DATA` with `ALGORITHM=BULK` supports the following data types:

- `INT`
- `SMALLINT`
- `TINYINT`
- `BIGINT`
- `CHAR`
- `BINARY`
- `VARCHAR`
- `VARBINARY`

- `NUMERIC`
- `DECIMAL`
- `UNSIGNED NUMERIC`
- `UNSIGNED DECIMAL`
- `DOUBLE`
- `FLOAT`
- `DATE`
- `DATETIME`
- `BIT`
- `ENUM`
- `JSON`
- `SET`
- `TIMESTAMP`
- `YEAR`
- `TINYBLOB`
- `BLOB`
- `MEDIUMBLOB`
- `LONGBLOB`
- `TINYTEXT`
- `TEXT`
- `MEDIUMTEXT`
- `LONGTEXT`
- `GEOMETRY`
- `GEOMETRYCOLLECTION`
- `POINT`
- `MULTIPOINT`
- `LINESTRING`
- `MULTILINESTRING`
- `POLYGON`
- `MULTIPOLYGON`
- `VECTOR`

Bulk Ingest Syntax

```
mysql> LOAD DATA
  [LOW_PRIORITY | CONCURRENT]
  [FROM]
  INFILE | URL | S3 'file_prefix' | 'options' [COUNT N]
  [IN PRIMARY KEY ORDER]
  INTO TABLE tbl_name
  [CHARACTER SET charset_name] [COMPRESSION = {'ZSTD'}]
  [{FIELDS | COLUMNS}
      [TERMINATED BY 'string']
      [[OPTIONALLY] ENCLOSED BY 'char']
      [ESCAPED BY 'char']
  ]
  [LINES
      [TERMINATED BY 'string']
  ]
  [IGNORE number {LINES | ROWS}]
  [PARALLEL = number]
  [MEMORY = M]
  [ALGORITHM = BULK]

options: {
   JSON_OBJECT("key","value"[,"key","value"] ...)
        "key","value": {
        "url-prefix","prefix"
        ["url-sequence-start",0]
        ["url-suffix","suffix"]
        ["url-prefix-last-append","@"]
        ["is-dryrun",{true|false}]
        }
}
```

The additional `LOAD DATA` clauses are:

- `FROM`: Makes the statement more readable.
- `URL`: A URL accessible with a HTTP GET request.
- `S3`: The AWS S3 file location.

  This requires the user privilege `LOAD_FROM_S3`.
- `COUNT`: The number of files in a series of files.

  For `COUNT 5` and *`file_prefix`* set to `data.csv.`, the five files would be: `data.csv.1`, `data.csv.2`, `data.csv.3`, `data.csv.4`, and `data.csv.5`.
- `IN PRIMARY KEY ORDER`: Use when the data is already sorted. The values should be in ascending order within the file.

  For a file series, the primary keys in each file must be disjoint and in ascending order from one file to the next.
- `PARALLEL`: The number of concurrent threads to use. A typical value might be 16, 32 or 48. The default value is 16.

  `PARALLEL` does not require `CONCURRENT`.
- `MEMORY`: The amount of memory to use. A typical value might be 512M or 4G. The default value is 1G.
- `ALGORITHM`: Set to `BULK` for bulk load. The file format is CSV.

- `COMPRESSION`: The file compression algorithm. Bulk load supports the ZSTD algorithm.
- `options` is a JSON object literal that includes:
  - `url-prefix`: The common URL prefix for the files to load.
  - `url-sequence-start`: The sequence number for the first file.

    The default value is 1, and the minimum value is 0. The value cannot be a negative number. The value can be a string or a number, for example, "134", or "default".
  - `url-suffix`: The file suffix.
  - `url-prefix-last-append`: The string to append to the prefix of the last file.

    This supports MySQL Shell dump files.
  - `is-dryrun`: Set to `true` to run basic checks and report if bulk load is possible on the given table. The default value is `false`.

    To enable `is-dryrun`, use any of the following values: `true`, `"true"`, `"1"`, `"on"` or `1`.

    To disable `is-dryrun`, use any of the following values: `false`, `"false"`, `"0"`, `"off"` or `0`.

`LOAD DATA` with `ALGORITHM=BULK` does not support these clauses:

```
LOAD DATA
  [LOCAL]
  [REPLACE | IGNORE]
  [PARTITION (partition_name [, partition_name] ...)]
  ]
  [LINES
      [STARTING BY 'string']
  ]
  [(col_name_or_user_var
      [, col_name_or_user_var] ...)]
  [SET col_name={expr | DEFAULT}
      [, col_name={expr | DEFAULT}] ...]
```

Syntax Examples

- An example that loads unsorted data from AWS S3 with 48 concurrent threads and 4G of memory:

```
mysql> GRANT LOAD_FROM_S3 ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM S3 's3-us-east-1://innodb-bulkload-dev-1/lineitem.tbl'
        INTO TABLE lineitem
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        PARALLEL = 48
        MEMORY = 4G
        ALGORITHM=BULK;
```

- An example that loads eight files of sorted data from AWS S3. The `file_prefix` ends with a period. The files are `lineitem.tbl.1`, `lineitem.tbl.2`, ... `lineitem.tbl.8`:

```
mysql> GRANT LOAD_FROM_S3 ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM S3 's3-us-east-1://innodb-bulkload-dev-1/lineitem.tbl.' COUNT 8
        IN PRIMARY KEY ORDER
        INTO TABLE lineitem
```

```
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        ALGORITHM=BULK;
```

- An example that performs a dry run on a sequence of MySQL Shell dump files compressed with the ZSTD algorithm:

```
mysql> GRANT LOAD_FROM_URL ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM URL
        '{"url-prefix","https://example.com/bucket/test@lineitem@","url-sequence-start",0,"url-suffix","
        COUNT 20
        INTO TABLE lineitem
        CHARACTER SET ???? COMPRESSION = {'ZSTD'}
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 20000 LINES
        ALGORITHM=BULK;
```

- An example that loads data with the URI keyword (supported as of MySQL 9.4.0):

```
mysql> GRANT LOAD_FROM_URL ON *.* TO load_user@localhost;

mysql> LOAD DATA FROM URI 'https://data_files.com/data_files_1.tbl'
        INTO TABLE lineitem
        FIELDS TERMINATED BY "|"
        OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        ALGORITHM=BULK;
```

- An example that monitors bulk load progress in a second session.
  - Review the list of stages with the following query:

```
mysql> SELECT NAME, ENABLED, TIMED FROM performance_schema.setup_instruments
        WHERE ENABLED='YES' AND NAME LIKE "stage/bulk_load%";
```

  - Enable the `events_stages_current` with the following query:

```
mysql> UPDATE performance_schema.setup_consumers
        SET ENABLED = 'YES' WHERE NAME LIKE 'events_stages_current';
```

  - Use one session to run bulk load, and monitor progress in a second session:

```
mysql> SELECT thread_id, event_id, event_name, WORK_ESTIMATED,  WORK_COMPLETED
        FROM performance_schema.events_stages_current;
--------------
SELECT thread_id, event_id, event_name, WORK_ESTIMATED,  WORK_COMPLETED FROM performance_schema.events_
--------------

+-----------+----------+----------------------------------+----------------+----------------+
| thread_id | event_id | event_name                       | WORK_ESTIMATED | WORK_COMPLETED |
+-----------+----------+----------------------------------+----------------+----------------+
|        49 |        5 | stage/bulk_load_unsorted/sorting |     1207551343 |      583008145 |
+-----------+----------+----------------------------------+----------------+----------------+
1 row in set (0.00 sec)
```

3.2 Bulk Ingest Data to MySQL Server Limitations

- `LOAD DATA` with `ALGORITHM=BULK` supports tables with at least one column with the `VECTOR` data type. If you attempt to load a table without at least one column with the `VECTOR` data type, an error occurs.

- `LOAD DATA` with `ALGORITHM=BULK` has the following limitations:
  - It locks the target table exclusively and does not allow other operations on the table.
  - It does not support automatic rounding or truncation of the input data. It fails if the input data requires rounding or truncation in order to be loaded.
  - It does not support temporary tables.
  - It is atomic but not transactional. It commits any transaction that is already running. On failure the `LOAD DATA` statement is completely rolled back.
  - It cannot execute when the target table is explicitly locked by a `LOCK TABLES` statement.
- The target table for `LOAD DATA` with `ALGORITHM=BULK` has the following limitations:
  - It must be empty. The state of the table should be as though it has been freshly created. If the table has instantly added/dropped column, call TRUNCATE before calling `LOAD DATA` with `ALGORITHM=BULK`.
  - It must not be partitioned.
  - It must not contain secondary indexes.
  - It must be in a file_per_tablespace, and must not be in a shared tablespace.
  - It must have the default row format, `ROW_FORMAT=DYNAMIC`. Use `ALTER TABLE` to make any changes to the table after `LOAD DATA` with `ALGORITHM=BULK`.
  - It must not contain virtual or stored generated columns.
  - It must not contain foreign keys.
  - It must not contain `CHECK` constraints.
  - It must not contain triggers.
  - It is not replicated to other nodes.

Chapter 4 Training and Using Machine Learning Models

Table of Contents

This chapter describes how to create and manage machine learning models with the AutoML feature of MySQL AI.

4.1 About AutoML

The AutoML feature of MySQL AI makes it easy to use machine learning (ML), whether you are a novice user or an experienced ML practitioner. You provide the data, and AutoML analyzes the characteristics of the data and creates an optimized machine learning model that you can use to generate predictions and explanations. An ML model makes predictions by identifying patterns in your data and applying those patterns to unseen data. AutoML explanations help you understand how these predictions are made, such as which features of a dataset contribute most to a prediction. You can score machine learning models to get a better understanding of the quality of the model and its ability to generate reliable predictions.

With AutoML, you can do the following:

- Classify Data
- Perform Regression Analysis
- Generate Forecasts
- Detect Anomalies
- Generate Recommendations
- Topic Modeling

4.1.1 AutoML Ease of Use

The AutoML feature of MySQL AI is purpose-built for ease of use. It requires no machine learning expertise, specialized tools, or algorithms. With AutoML and a set of training data, you can train a predictive machine learning model with a single call to the `ML_TRAIN` SQL routine.

For example:

```
CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', NULL, @census_model);
```

The `ML_TRAIN` routine leverages Oracle AutoML technology to automate training of machine learning models. Learn more about Oracle AutoML.

You can use a model created by `ML_TRAIN` with other AutoML routines to generate predictions and explanations. For example, the following call to the `ML_PREDICT_TABLE` routine generates predictions for a table of input data:

```
CALL sys.ML_PREDICT_TABLE('heatwaveml_bench.census_test', @census_model,
'heatwaveml_bench.census_predictions', NULL);
```

All AutoML operations are initiated by running `CALL` or `SELECT` statements, which can be easily integrated into your applications. AutoML routines reside in the MySQL `sys` schema. Learn more about AutoML Routines.

In addition, with AutoML, there is no need to move or reformat your data, which saves you time and effort while keeping your data and models secure.

To start using AutoML with sample datasets, see Machine Learning Use Cases.

What's Next

- Learn more about the following:
  - AutoML Supervised Learning
  - AutoML Workflow
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.

4.1.2 AutoML Workflow

A typical AutoML workflow is described below:

1. When you run the ML_TRAIN routine, AutoML retrieves the data to use for training. The training data is then distributed across the cluster, which performs machine learning computation in parallel. See Train a Model.
2. AutoML analyzes the training data, trains an optimized machine learning model, and stores the model in a model catalog. See Model Catalog.
3. AutoML ML_PREDICT_* and ML_EXPLAIN_* routines use the trained model to generate predictions and explanations on test or unseen data. See Generate Predictions and Generate Explanations.
4. Predictions and explanations are returned to the user or application that issued the query.

Optionally, the ML_SCORE routine can be used to compute the quality of a model to ensure that predictions and explanations are reliable. See Score a Model.

To start using AutoML with sample datasets, see Machine Learning Use Cases.

What's Next

- Learn more about the following:
  - AutoML Learning Types
  - AutoML Ease of Use
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.

4.1.3 AutoML Learning Types

AutoML supports the following types of machine learning: supervised, unsupervised, and semi-supervised.

Supervised Learning

Supervised learning creates a machine learning model by analyzing a labeled dataset to learn patterns. This means that the dataset has values associated with the column (the label) that the machine learning model eventually generates predictions for. The model is able to predict labels based on the features of the dataset. For example, a census and income dataset may have features such as age, education, occupation, and country that you can use to predict the income of an individual (the label). The income label in this dataset already has values that the machine learning model uses for training.

Once a machine learning model is trained, it can be used on unseen data, where the label is unknown, to make predictions. In a business setting, predictive models have a variety of possible applications such as predicting customer churn, approving or rejecting credit applications, predicting customer wait times, and so on.

See Labeled Data and Unlabeled Data to learn more.

Unsupervised Learning

Unsupervised learning is available for forecasting, anomaly detection and topic modeling use cases. This type of learning requires no labeled data. This means that the column (the label) the machine learning model eventually generates predictions for has no values in the dataset for training. For example, a dataset of credit card transactions that you use for anomaly detection has a column indicating if the transaction is anomalous or normal, but the column has no data (unlabeled). See Generate Forecasts, Detect Anomalies, and Topic Modeling to learn more.

Semi-Supervised Learning

Semi-supervised learning for anomaly detection uses a specific set of labeled data along with unlabeled data to detect anomalies. The dataset for this type of model must have a column whose only allowed values are 0 (normal), 1, (anomalous), and NULL (unlabeled). All rows in the dataset are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component. See Detect Anomalies and Anomaly Detection Model Types to learn more.

What's Next

- Learn more about the following:
  - AutoML Ease of Use
  - AutoML Workflow
  - Oracle AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.1.4 Oracle AutoML

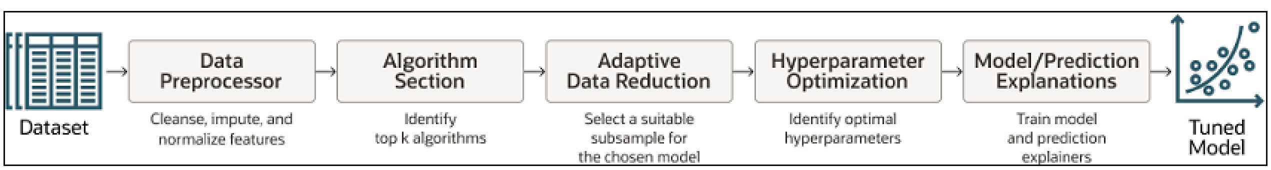
The AutoML `ML_TRAIN` routine leverages Oracle AutoML technology to automate the process of training a machine learning model. Oracle AutoML replaces the laborious and time consuming tasks of the data analyst, whose workflow is as follows:

1. Selecting a model from a large number of viable candidate models.
2. For each model, tuning hyperparameters.
3. Selecting only predictive features to speed up the pipeline and reduce over-fitting.
4. Ensuring the model performs well on unseen data (also called generalization).

Oracle AutoML automates this workflow, providing you with an optimal model given a time budget. When you run the AutoML `ML_TRAIN` routine, that triggers the Oracle AutoML pipeline to run the following stages in a single command:

- Data pre-processing
- Algorithm selection
- Adaptive data reduction
- Hyperparameter optimization
- Model and prediction explanations

Figure 4.1 Oracle AutoML Pipeline



Oracle AutoML also produces high quality models very efficiently, which is achieved through a scalable design and intelligent choices that reduce trials at each stage in the pipeline.

- *Scalable design*: The Oracle AutoML pipeline is able to exploit both MySQL AI internode and intranode parallelism, which improves scalability and reduces runtime.
- *Intelligent choices reduce trials in each stage*: Algorithms and parameters are chosen based on dataset characteristics, which ensures that the model is accurate and efficiently selected. This is achieved using meta-learning throughout the pipeline.

For additional information about Oracle AutoML, refer to Yakovlev, Anatoly, et al. "Oracle AutoML: A Fast and Predictive AutoML Pipeline." Proceedings of the VLDB Endowment 13.12 (2020): 3166-3180.

What's Next

- Learn more about the following:
  - AutoML Learning Types
  - AutoML Ease of Use
  - AutoML Workflow
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.2 Additional AutoML Requirements

Before You Begin

Model and Table Sizes

- The table used to train a model cannot exceed 10 GB, 100 million rows, or 1017 columns.

Data Requirements

- Each dataset must reside in a single table on the MySQL server. AutoML routines operate on a single table.
- Table columns must use supported data types. See Supported Data Types for AutoML to learn more.
- NaN (Not a Number) values are not recognized by MySQL and should be replaced by `NULL`.
- Refer to the following requirements for specific machine learning models.
  - Classification models: Must have at least two distinct values, and each distinct value should appear in at least five rows.
  - Regression models: The target column must be numeric.

Note

The `ML_TRAIN` routine ignores columns missing more than 20% of its values and columns with the same value in each row. Missing values in numerical columns are replaced with the average value of the column, standardized to a mean of 0 and with a standard deviation of 1. Missing values in categorical columns are replaced with the most frequent value, and either one-hot or ordinal encoding is used to convert categorical values to numeric values. The input data as it exists in the MySQL database is not modified by `ML_TRAIN`.

MySQL User Names

To use AutoML, ensure that the MySQL user name that trains a model does not have a period character ("."). For example, a user named `'joesmith'@'%'` is permitted to train a model, but a user named `'joe.smith'@'%'` is not. The model catalog schema created by the `ML_TRAIN` procedure incorporates the user name in the schema name (for example, `ML_SCHEMA_joesmith`), and a period is not a permitted schema name character.

What's Next

- Learn more about the following:
  - AutoML Privileges
  - Supported Data Types for AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.3 AutoML Privileges

To use AutoML, ask the admin user to grant you the following privileges. Replace *`user_name`* and *`database_name`* in the commands with the appropriate user name and database name.

Database Privileges

You need the following privileges to access the database that stores the input tables (training datasets).

```
mysql> GRANT SELECT, ALTER ON database_name.* TO 'user_name'@'%';
```

You need the following privileges to access the database that stores the output tables of generated predictions and explanations.

```
mysql> GRANT CREATE, DROP, INSERT, SELECT, ALTER, DELETE, UPDATE ON database_name.* TO 'user_name'@'%';
```

Tracking and Monitoring Privileges

You need the following privileges to track/monitor the status of AutoML and AutoML routines..

```
mysql> GRANT SELECT ON performance_schema.rpd_tables TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_table_id TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_query_stats TO 'user_name'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_ml_stats TO 'user_name'@'%';
```

Model Catalog Privileges

You need the following privileges to access machine learning models from the model catalog.

```
mysql> GRANT SELECT, INSERT, CREATE, ALTER, UPDATE, DELETE, DROP, GRANT OPTION ON ML_SCHEMA_user_name.* TO 'us
```

System Privileges

You need the following privileges for the system database where MySQL AI routines reside.

```
mysql> GRANT SELECT, EXECUTE ON sys.* TO 'user_name'@'%';
```

What's Next

- Learn more about the following:
  - Additional AutoML Requirements
  - Supported Data Types for AutoML
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.4 Supported Data Types for AutoML

AutoML supports the following data types.

Numeric Data Types

- `DECIMAL`
- `DOUBLE`
- `FLOAT`
- `INT`
- `INT UNSIGNED`
- `TINYINT`
- `TINYINT UNSIGNED`
- `SMALLINT`
- `SMALLINT UNSIGNED`
- `MEDIUMINT`
- `MEDIUMINT UNSIGNED`
- `BIGINT`
- `BIGINT UNSIGNED`

Temporal Data Types

- `DATE`
- `TIME`
- `DATETIME`
- `TIMESTAMP`
- `YEAR`

String and Text Data Types

- `VARCHAR`

- CHAR
- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

Data Type Limitations

AutoML uses TfidfVectorizer to pre-process TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT, and appends the results to the data set. AutoML has the following limitations for text usage:

- The ML_PREDICT_TABLE ml_results column contains the prediction results and the data. This combination must be fewer than 65,532 characters.
- AutoML only supports datasets in the English language.
- AutoML does not support text columns with NULL values.
- AutoML does not support a text target column.
- AutoML does not support recommendation tasks with a text column.
- For the forecasting task, endogenous_variables cannot be text.

What's Next

- Learn more about the following:
  - Additional AutoML Requirements
  - AutoML Privileges
- Learn how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.5 Creating a Machine Learning Model

The topics in this section go through the process of training and using a machine learning model.

Before going through these tasks, make sure to Review Additional AutoML Requirements.

To start using AutoML with sample datasets, see Machine Learning Use Cases.

4.5.1 Preparing Data

AutoML works with labeled and unlabeled data to train and score machine learning models.

Before You Begin

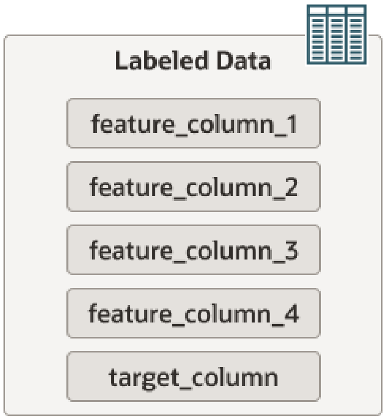
- Review the Requirements.
- Get the Required Privileges to use AutoML.

- Review the Data Types Supported For Machine Learning Tasks.

Labeled Data

Labeled data is data that has values associated with it. It has feature columns and a target column (the *label*), as illustrated in the following diagram:

Figure 4.2 Labeled Data



Feature columns contain the input variables used to train the machine learning model. The target column contains *ground truth values* or, in other words, the correct answers. This dataset can be considered the *training dataset*.

A labeled dataset with ground truth values is also used to score a model (compute its accuracy and reliability). This dataset should have the same columns as the *training dataset* but with a different set of data. This dataset can be considered the *validation dataset*.

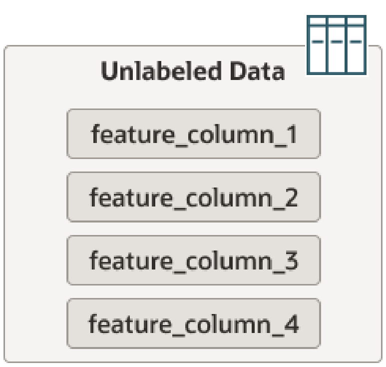
Labeled Data Example

A table of data for bank customers can be a labeled dataset. The feature columns in the table have data related to job, marital status, education, and city of residence. The target column has the approval status of a loan application, `Yes` or `No`. You can use some of the data in this table to train a classification machine learning model. You can also use the data in the table that wasn't used for training to score the trained machine learning model.

Unlabeled Data

Unlabeled data has feature columns but no target column (no answers), as illustrated below:

Figure 4.3 Unlabeled Data



If you are training a machine learning model that does not require labeled data, such as models for topic modeling or anomaly detection, you use unlabeled data. AutoML also uses unlabeled data to generate predictions and explanations. It must have exactly the same feature columns as the training dataset but no target column. This type of dataset can be considered the *test dataset*. Test data starts as labeled data, but the label is not considered when the machine learning model generates predictions and explanations. This allows you to compare the generated predictions and explanations with the real values in the dataset before you start using “unseen data”.

The “unseen data” that you eventually use with your model to make predictions is also unlabeled data. Like the *test dataset*, unseen data must have exactly the same feature columns as the training dataset but no target column.

Unlabeled Data Example

A table of data for credit card transactions can be an unlabeled dataset. The feature columns in the table have data related to the amount of the purchase and the location of the purchase. Because there is no column identifying any transactions as anomalous or normal, it is unlabeled data. AutoML can train an anomaly detection model on the unlabeled data to try and find unusual patterns in the data. A different set of labeled data identifying anomalies in credit cards transactions can be used to score the trained model.

Example Datasets

To start using AutoML with sample datasets, see Machine Learning Use Cases. Alternatively, navigate to the *AutoML examples and performance benchmarks* GitHub repository at https://github.com/oracle-samples/heatwave-ml.

What's Next

- Learn how to Train a Model.

4.5.2 Training a Model

Run the `ML_TRAIN` routine on a training dataset to produce a trained machine learning model.

Before You Begin

- Review how to Prepare Data.
- Review Additional AutoML Requirements.

ML_TRAIN Overview

`ML_TRAIN` supports training of the following models:

- Classification: Assign items to defined categories.
- Regression: Generate a prediction based on the relationship between a dependent variable and one or more independent variables.
- Forecasting: Use a timeseries dataset to generate forecasting predictions.
- Anomaly Detection: Detect unusual patterns in data.
- Recommendation: Generate user and product recommendations.

- Topic Modeling: Generate words and similar expressions that best characterize a set of documents.

The training dataset used with `ML_TRAIN` must reside in a table on the MySQL server.

`ML_TRAIN` stores machine learning models in the `MODEL_CATALOG` table. See The Model Catalog to learn more.

The time required to train a model can take a few minutes to a few hours depending on the following:

- The number of rows and columns in the dataset. AutoML supports tables up to 10 GB in size with a maximum of 100 million rows and or 1017 columns.
- The specified `ML_TRAIN` parameters.

To learn more about `ML_TRAIN` requirements and options, see ML_TRAIN or Machine Learning Use Cases.

The quality and reliability of a trained model can be assessed using the `ML_SCORE` routine. For more information, see Score a Model. `ML_TRAIN` displays the following message if a trained model has a low score: `Model Has a low training score, expect low quality model explanations`.

ML_TRAIN Example

Before training a model, it is good practice to define your own model handle instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates. See Defining Model Handle to learn more.

To train a machine learning model:

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @census_model = 'census_test';
   ```

   The model handle is set to `census_test`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @variabl
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`variable`* with your own values.

   The following example runs `ML_TRAIN` on the `census_data.census_train` training dataset.

   ```
   mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'),
   ```

   Where:

   - `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
   - `revenue` is the name of the target column, which contains ground truth values.
   - `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.

- `@census_model` is the session variable previously set that defines the model handle to the name defined by the user: `census_test`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training completes, query the model catalog for the model handle and the name of the trained table to confirm the model handle is correctly set. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+------------------------------------------------------+----------------------------------+
| model_handle                                         | train_table_name                 |
+------------------------------------------------------+----------------------------------+
| census_test                                          | census_data.census_train         |
+------------------------------------------------------+----------------------------------+
1 row in set (0.0450 sec)
```

Tip

When done working with a trained model, it is good practice to unload it. See Unload a Model.

What's Next

- For details on all training options and to view more examples for task-specific models, see ML_TRAIN.
- Learn how to Load a Model.

4.5.3 Loading a Model

You must load a machine learning model from the model catalog before running AutoML routines other than `ML_TRAIN`. A model remains loaded and can be called repetitively by AutoML routines until it is unloaded using the `ML_MODEL_UNLOAD` routine, or until the cluster is restarted.

A model can only be loaded by the MySQL user that created the model unless you grant access to other users. For more information, see Grant Other Users Access to a Model.

Review ML_MODEL_LOAD parameter descriptions.

Before You Begin

- Review how to Train a Model.

Loading a Model with the Session Variable

After training a model, you set a session variable for the model handle that you can use until the current connection ends.

The following example loads an AutoML model from the model catalog by using the session variable

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

Where:

- `@census_model` is the session variable that contains the model handle.

- NULL is specified in place of the user name of the model owner. You are not required to specify a user name.

Loading a Model Handle with the Defined Model Handle Name

Before training a machine learning model, it is good practice to define a model handle name instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates. See ML_TRAIN Example.

The following example uses the defined model handle name to load the model.

```
mysql> CALL sys.ML_MODEL_LOAD('census_test', NULL);
```

Loading the Model with the Automatically Generated Model Handle

If you do not define a model handle name before training a machine learning model, it is automatically generated. If the connection for the session variable of a model handle ends, you need to load the model with the model name.

1. Query the model handle, model owner, and the trained table name from the model catalog table. Replace user1 with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+------------------------------------------------------+-------------+--------------------------------+
| model_handle                                         | model_owner | train_table_name               |
+------------------------------------------------------+-------------+--------------------------------+
| census_data.census_train_admin_1745261646953         | admin       | census_data.census_train       |
| census_data.census_train_admin_1745334557047         | admin       | census_data.census_train       |
| census_data.census_train_admin_1745336500455         | admin       | census_data.census_train       |
+------------------------------------------------------+-------------+--------------------------------+
3 rows in set (0.0431 sec)
```

2. Copy the appropriate model_handle and use it to load the machine learning model.

```
mysql> CALL sys.ML_MODEL_LOAD('census_data.census_train_user1_1745261646953', NULL);
```

Verifying Model is Loaded

You have the option to verify that model is loaded by using the ML_MODEL_ACTIVE routine.

The following example verifies the model previously loaded is active.

1. Run ML_MODEL_ACTIVE on all active and loaded models and assign a session variable.

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @variable);
```

Replace *variable* with your own value. For example:

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @models);
```

2. Query the session variable previously created. Replace *models* with your own value.

```
mysql> SELECT @models;
+-----------------------------------------------------------------------------------------------------
| @models
+-----------------------------------------------------------------------------------------------------
| [{"total model size(bytes)": 388948}, {"admin": [{"census_test": {"format": "HWMLv2.0", "model_size(b
+-----------------------------------------------------------------------------------------------------
```

```
1 row in set (0.0431 sec)
```

The output displays the loaded model with information on the user that trained the model, the size of the model, the model handle, and its format.

What's Next

- For details on all model load options, see ML_MODEL_LOAD.
- Learn how to Generate Predictions.

4.5.4 Generating Predictions

Predictions are generated by running `ML_PREDICT_ROW` or `ML_PREDICT_TABLE` on trained models. The row or table of data must have the same feature columns as the data used to train the model. If the target column exists in the data to run predictions on, it is not considered during prediction generation.

`ML_PREDICT_ROW` generates predictions for one or more rows of data. `ML_PREDICT_TABLE` generates predictions for an entire table of data and saves the results to an output table.

4.5.4.1 Generating Predictions for a Row of Data

`ML_PREDICT_ROW` generates predictions for one or more rows of data specified in `JSON` format. You invoke the routine with the `SELECT` statement.

This topic has the following sections.

- Before You Begin
- Preparing to Generate a Row Prediction
- Inputting Data to Generate a Row Prediction
- Generating Predictions on One or More Rows of Data
- What's Next

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Preparing to Generate a Row Prediction

Before running `ML_PREDICT_ROW`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

   ```
   mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cer
   ```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate predictions on one or more rows of data. For parameter and option descriptions, see ML_PREDICT_ROW.

Inputting Data to Generate a Row Prediction

One way to generate predictions on row data is to manually enter the row data into a session variable, and then generate a prediction by specifying the session variable.

1. Define values for each column to predict. The column names must match the feature column names in the trained table.

   ```
   mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle, opt
   ```

   In the following example, create the `@row_input` session variable and enter the data to predict into the session variable.

   ```
   mysql> SET @row_input = JSON_OBJECT(
             "age", 25,
             "workclass", "Private",
             "fnlwgt", 226802,
             "education", "11th",
             "education-num", 7,
             "marital-status", "Never-married",
             "occupation", "Machine-op-inspct",
             "relationship", "Own-child",
             "race", "Black",
             "sex", "Male",
             "capital-gain", 0,
             "capital-loss", 0,
             "hours-per-week", 40,
             "native-country", "United-States");
   ```

2. Run `ML_PREDICT_ROW` and specify the session variable set previously. Optionally, use `\G` to display information in an easily readable format.

   ```
   mysql> SELECT sys.ML_PREDICT_ROW(@variable, ...), model_handle, options);
   ```

   Replace *`variable`*, *`model_handle`*, and *`options`* with your own values. For example:

   ```
   mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @census_model, NULL)\G
   *************************** 1. row ***************************
   sys.ML_PREDICT_ROW(@row_input, @census_model, NULL):
   {
       "age": 25,
       "sex": "Male",
       "race": "Black",
       "fnlwgt": 226802,
       "education": "11th",
       "workclass": "Private",
       "Prediction": "<=50K",
       "ml_results": {
           "predictions": {
               "revenue": "<=50K"
           },
           "probabilities": {
               ">50K": 0.0032,
               "<=50K": 0.9968
           }
       },
   ```

```
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States"
}
1 row in set (2.2218 sec)
```

Where:

- `@row_input` is a session variable containing a row of unlabeled data. The data is specified in `JSON` key-value format. The column names must match the feature column names in the training dataset.
- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.
- `NULL` sets no options to the routine.

The prediction on the data is that the revenue is `<=50K with a probability of 99.7%.`.

Generating Predictions on One or More Rows of Data

Another way to generate predictions is to create a `JSON_OBJECT` with specified columns and labels, and then generate predictions on one or more rows of data in the table.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_name", sc
        FROM input_table_name LIMIT N;
```

The following example specifies the table and columns to use for the prediction and assigns output labels for each table-column pair. No options are set with `NULL`. It also defines to predict the top two rows of the table. Optionally, use `\G` to display information in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, NULL)FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                            "age": 62,
                                            "sex": "Female",
                                            "race": "White",
                                            "fnlwgt": 123582,
```

```
                                            "education": "10th",
                                            "workclass": "Private",
                                            "Prediction": "<=50K",
                                            "ml_results": {
                                                "predictions": {
                                                    "revenue": "<=50K"
                                                },
                                                "probabilities": {
                                                    ">50K": 0.0106,
                                                    "<=50K": 0.9894
                                                }
                                            },
                                            "occupation": "Other-service",
                                            "capital-gain": 0,
                                            "capital-loss": 0,
                                            "relationship": "Unmarried",
                                            "education-num": 6,
                                            "hours-per-week": 40,
                                            "marital-status": "Divorced",
                                            "native-country": "United-States"
                                        }
*************************** 2. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                            "age": 32,
                                            "sex": "Female",
                                            "race": "White",
                                            "fnlwgt": 174215,
                                            "education": "Bachelors",
                                            "workclass": "Federal-gov",
                                            "Prediction": "<=50K",
                                            "ml_results": {
                                                "predictions": {
                                                    "revenue": "<=50K"
                                                },
                                                "probabilities": {
                                                    ">50K": 0.3249,
                                                    "<=50K": 0.6751
                                                }
                                            },
                                            "occupation": "Exec-managerial",
                                            "capital-gain": 0,
                                            "capital-loss": 0,
                                            "relationship": "Not-in-family",
                                            "education-num": 13,
                                            "hours-per-week": 60,
                                            "marital-status": "Never-married",
                                            "native-country": "United-States"
                                        }
2 rows in set (9.6548 sec)
```

The output generates revenue predictions for the four rows of data.

What's Next

- Review ML_PREDICT_ROW for parameter descriptions and options.
- After generating predictions for a row of data, learn how to Generate Explanations for a Row of Data to get insight into which features have the most influence on the predictions.
- Learn how to Generate Predictions for a Table.

- Learn how to Score a Model to get insight into the quality of the model.

4.5.4.2 Generating Predictions for a Table

`ML_PREDICT_TABLE` generates predictions for an entire table of trained data. Predictions are performed in parallel.

`ML_PREDICT_TABLE` is a compute intensive process. If `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Input Tables and Output Tables

You can specify the output table and the input table as the same table if all the following conditions are met:

- The input table does not have the columns that are created for the output table when generating predictions. Output columns are specific to each machine learning task. Some of these columns include:
  - `Prediction`
  - `ml_results`
- The input table does not have a primary key, and it does not have a column named `_4aad19ca6e_pk_id`. This is because `ML_PREDICT_TABLE` adds a column as the primary key with the name `_4aad19ca6e_pk_id` to the output table.
- The input table was not trained with the `log_anomaly_detection` task.

If you specify the output table and the input table as the same name, the predictions are inserted into the input table.

Preparing to Generate Predictions for a Table

Before running `ML_PREDICT_TABLE`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cen
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate predictions for a table of data. For parameter and option descriptions, see ML_PREDICT_TABLE.

Generating Predictions for a Table

To generate predictions for a table, define the input table, the model handle, the output table, and any additional options.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

The following example generates predictions for the entire table in the trained and loaded model.

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.census_train', @census_model, 'census_data.census_train_predi
```

Where:

- `census_data.census_train` is the fully qualified name of the test dataset table (*`schema_name.table_name`*). The table must have the same feature column names as the training dataset. The target column is not required. If it present in the table, it is not considered when generating predictions.

- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.

- `census_data.census_train_predictions` is the output table where predictions are stored. A fully qualified table name must be specified (*`schema_name.table_name`*). If the table already exists, an error is returned.

- `NULL` sets no options to the routine.

When the output table is created, you can query a sample of the table to review predictions.

```
mysql> SELECT * FROM table_name LIMIT N;
```

Replace *`table_name`* with your own table name, and *`N`* with the number of rows from the table you want to view.

The following example queries the top five rows of the output table.

```
mysql> SELECT * FROM census_train_predictions LIMIT 5;
+-------------------+-----+------------------+--------+--------------+----------------+---------------------
| _4aad19ca6e_pk_id | age | workclass        | fnlwgt | education    | education-num | marital-status
+-------------------+-----+------------------+--------+--------------+----------------+---------------------
|                 1 |  37 | Private          |  99146 | Bachelors    |             13 | Married-civ-spouse
|                 2 |  34 | Private          |  27409 | 9th          |              5 | Married-civ-spouse
|                 3 |  30 | Private          | 299507 | Assoc-acdm   |             12 | Separated
|                 4 |  62 | Self-emp-not-inc | 102631 | Some-college |             10 | Widowed
|                 5 |  51 | Private          | 153486 | Some-college |             10 | Married-civ-spouse
+-------------------+-----+------------------+--------+--------------+----------------+---------------------
5 rows in set (0.0014 sec)
```

The predictions and associated probabilities are displayed in the `ml_results` column. You can compare the predicted revenue values with the real revenue values in the table. If needed, you can refine and train different sets of data to try and generate more reliable predictions.

What's Next

- Review ML_PREDICT_TABLE for parameter descriptions and options.

- After generating predictions on a table, learn how to Generate Explanations on a table to get insights into which features have the most influence on the predictions.

- Learn how to Generate Predictions for a Row of Data.
- Learn how to Score a Model to get insight into the quality of the model.

4.5.5 Generating Model Explanations

After the `ML_TRAIN` routine, use the `ML_EXPLAIN` routine to train model explainers for AutoML. By default, the `ML_TRAIN` routine trains the Permutation Importance model explainer.

This topic has the following sections.

- Before You Begin
- Explanations Overview
- Model Explainers
- Unsupported Model Types
- Preparing to Generate a Model Explanation
- Retrieve the Default Permutation Importance Explanation
- Generating a Model Explanation
- What's Next

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Explanations Overview

Explanations help you understand which features have the most influence on a prediction. Feature importance is presented as a value ranging from -1 to 1. A positive value indicates that a feature contributed toward the prediction. A negative value indicates that the feature contributed toward a different prediction. For example, if a feature in a loan approval model with two possible predictions ('approve' and 'reject') has a negative value for an 'approve' prediction, that feature would have a positive value for a 'reject' prediction. A value of 0 or near 0 indicates that the feature value has no impact on the prediction to which it applies.

Model Explainers

Model explainers are used when you run the `ML_EXPLAIN` routine to explain what the model learned from the training dataset. The model explainer provides a list of feature importance to show what features the model considered important based on the entire training dataset. The `ML_EXPLAIN` routine can train these model explainers:

- The Permutation Importance model explainer, specified as `permutation_importance`, is the default model explainer. `ML_TRAIN` generates this model explainer when it runs.

- The Partial Dependence model explainer, specified as `partial_dependence`, shows how changing the values of one or more columns changes the value that the model predicts. When you train this model explainer, you need to specify some additional options. See ML_EXPLAIN to learn more.
- The SHAP model explainer, specified as `shap`, produces feature importance values based on Shapley values.
- The Fast SHAP model explainer, specified as `fast_shap`, is a subsampling version of the SHAP model explainer, which usually has a faster runtime.

The model explanation is stored in the model catalog along with the machine learning model in the `model_explanation` column. See The Model Catalog. If you run `ML_EXPLAIN` again for the same model handle and model explainer, the field is overwritten with the new result.

Unsupported Model Types

You cannot generate model explanations for the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

Preparing to Generate a Model Explanation

Before running `ML_EXPLAIN`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'),
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate model explanations. For option and parameter descriptions, see ML_EXPLAIN.

Retrieve the Default Permutation Importance Explanation

After training and loading a model, you can retrieve the default model explanation using the `permutation_importance` explainer from the model catalog. See The Model Catalog.

```
mysql> SELECT column FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=model_handle;
```

The following example retrieves the model explainer column from the model catalog of the previously trained model. The `JSON_PRETTY` parameter displays the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census_
+--------------------------------------------------------------------------------------------------------
```

```
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0292,
    "sex": 0.0023,
    "race": 0.0019,
    "fnlwgt": 0.0038,
    "education": 0.0008,
    "workclass": 0.0068,
    "occupation": 0.0223,
    "capital-gain": 0.0479,
    "capital-loss": 0.0117,
    "relationship": 0.0234,
    "education-num": 0.0352,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------
1 row in set (0.0427 sec)
```

Replace `user1` and `@census_model` with your own user name and session variable.

The explanation displays values of permutation importance for each column.

Generating a Model Explanation

To generate a model explanation, run the `ML_EXPLAIN` routine.

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
```

The following example generates a model explanation on the trained and loaded model with the `shap` model explainer.

```
mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('model_explainer'
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `@census_model` is the session variable for the trained model.
- `model_explainer` is set to `shap` for the SHAP model explainer.

After running `ML_EXPLAIN`, you can view the model explanation in the Model Catalog. See The Model Catalog. The following example views the model explanation for the previous command. It provides values for each column representing importance values with the `shap` explainer.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census_mod
+-----------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------
| {
  "shap": {
    "age": 0.0467,
    "sex": 0.033,
```

```
    "race": 0.0155,
    "fnlwgt": 0.0185,
    "education": 0.016,
    "workclass": 0.0255,
    "occupation": 0.0001,
    "capital-gain": 0.0217,
    "capital-loss": 0.0001,
    "relationship": 0.0426,
    "education-num": 0.0186,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  },
  "permutation_importance": {
    "age": -0.0057,
    "sex": 0.0002,
    "race": 0.0001,
    "fnlwgt": 0.0103,
    "education": 0.0108,
    "workclass": 0.0189,
    "occupation": 0.0,
    "capital-gain": 0.0304,
    "capital-loss": 0.0,
    "relationship": 0.0195,
    "education-num": 0.0152,
    "hours-per-week": 0.0235,
    "marital-status": 0.0099,
    "native-country": 0.0
   }
} |
+------------------------------------------------------------------------------------------------------
1 row in set (0.0427 sec)
```

What's Next

- Review ML_EXPLAIN for parameter descriptions and options.
- Learn how to Generate Prediction Explanations.
- Learn more about the The Model Catalog.

4.5.6 Generating Prediction Explanations

Prediction explanations are generated by running `ML_EXPLAIN_ROW` or `ML_EXPLAIN_TABLE` on unlabeled data. The data must have the same feature columns as the data used to train the model. The target column is not required.

Prediction explanations are similar to model explanations, but rather than explain the whole model, prediction explanations explain predictions for individual rows of data. See Explanations Overview to learn more.

You can train the following prediction explainers:

- The Permutation Importance prediction explainer, specified as `permutation_importance`, is the default prediction explainer, which explains the prediction for a single row or table. Right after training and loading a model, you can run `ML_EXPLAIN_ROW` and `ML_EXPLAIN_TABLE` with this prediction explainer directly without having to run `ML_EXPLAIN` first.
- The SHAP prediction explainer, specified as `shap`, uses feature importance values to explain the prediction for a single row or table. To run this prediction explainer with `ML_EXPLAIN_ROW` and `ML_EXPLAIN_TABLE`, you must run `ML_EXPLAIN` first.

ML_EXPLAIN_ROW generates explanations for one or more rows of data. ML_EXPLAIN_TABLE generates explanations on an entire table of data and saves the results to an output table. ML_EXPLAIN_* routines limit explanations to the 100 most relevant features.

4.5.6.1 Generating Prediction Explanations for a Row of Data

ML_EXPLAIN_ROW explains predictions for one or more rows of unlabeled data. You invoke the routine by using a SELECT statement.

This topic has the following sections.

- Before You Begin
- Unsupported Model Types
- Preparing to Generate a Row Explanation
- Generating a Row Prediction Explanation with the Default Permutation Importance Explainer
- Generating a Row Prediction Explanation with the SHAP Explainer
- What's Next

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Unsupported Model Types

You cannot generate prediction explanations on a row of data for the following model types:

- Forecasting
- Recommendation
- Anomaly detection
- Anomaly detection for logs
- Topic modeling

Preparing to Generate a Row Explanation

Before running ML_EXPLAIN_ROW, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

   ```
   mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cen
   ```

2. The following example loads the trained model.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
   ```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate prediction explanations for one or more rows. For parameter and option descriptions, see ML_EXPLAIN_ROW.

Generating a Row Prediction Explanation with the Default Permutation Importance Explainer

After training and loading a model, you can run `ML_EXPLAIN_ROW` to generate a row prediction explanation with the default Permutation Importance explainer. However, if you train the `shap` prediction explainer with `ML_EXPLAIN`, you need to run `ML_EXPLAIN` again with the `permutation_importance` explainer before running `ML_EXPLAIN_ROW` with the same explainer.

The following example enters a row of data to explain into a session variable. The session variable is then used in the `ML_EXPLAIN_ROW` routine.

1. Define values for each column to predict. The column names must match the feature column names in the trained table.

   ```
   mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle, opt
   ```

   In the following example, assign the data to analyze into the `@row_input` session variable.

   ```
   mysql> SET @row_input = JSON_OBJECT(
             "age", 31,
             "workclass", "Private",
             "fnlwgt", 45781,
             "education", "Masters",
             "education-num", 14,
             "marital-status", "Married-civ-spouse",
             "occupation", "Prof-specialty",
             "relationship", "Not-in-family",
             "race", "White",
             "sex", "Female",
             "capital-gain", 14084,
             "capital-loss", 2042,
             "hours-per-week", 40,
             "native-country", "India");
   ```

2. Run the `ML_EXPLAIN_ROW` routine.

   ```
   mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);
   ```

   In the following example, include the session variable previously created. Optionally, use `\G` to display the output in an easily readable format. The output is similar to the following:

   ```
   mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'permut
   *************************** 1. row ***************************
   sys.ML_EXPLAIN_ROW(@row_input, @census_model,
             JSON_OBJECT('prediction_explainer', 'permutation_importance')):
             {
                   "age": 31,
                   "sex": "Female",
                   "race": "White",
                   "Notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                   "fnlwgt": 45781,
                   "education": "Masters",
                   "workclass": "Private",
                   "Prediction": ">50K",
                   "ml_results": {
                      "notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                      "predictions": {
                          "revenue": ">50K"
                       },
                       "attributions": {
                           "age": 0.34,
   ```

```
                        "sex": 0,
                        "race": 0,
                        "fnlwgt": 0,
                        "education": 0,
                        "workclass": 0,
                        "occupation": 0,
                        "capital-gain": 0.97,
                        "capital-loss": 0,
                        "relationship": 0,
                        "education-num": 0.04,
                        "hours-per-week": 0,
                        "marital-status": 0
                    }
                },
                "occupation": "Prof-specialty",
                "capital-gain": 14084,
                "capital-loss": 2042,
                "relationship": "Not-in-family",
                "education-num": 14,
                "hours-per-week": 40,
                "marital-status": "Married-civ-spouse",
                "native-country": "India",
                "age_attribution": 0.34,
                "sex_attribution": 0,
                "race_attribution": 0,
                "fnlwgt_attribution": 0,
                "education_attribution": 0,
                "workclass_attribution": 0,
                "occupation_attribution": 0,
                "capital-gain_attribution": 0.97,
                "capital-loss_attribution": 0,
                "relationship_attribution": 0,
                "education-num_attribution": 0.04,
                "hours-per-week_attribution": 0,
                "marital-status_attribution": 0
          }
1 row in set (6.3072 sec)
```

The output provides an explanation on the column that had the largest impact towards the prediction, and the column that contributed the most against the prediction.

Generating a Row Prediction Explanation with the SHAP Explainer

To generate a row prediction explanation with the SHAP explainer, you must first run the SHAP explainer with `ML_EXPLAIN`.

1. Run the `ML_EXPLAIN` routine.

   ```
   mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
   ```

   The following example runs the `shap` explainer.

   ```
   mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('prediction_ex
   ```

   Where:

   - `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
   - `revenue` is the name of the target column, which contains ground truth values.
   - `@census_model` is the session variable for the trained model.
   - `prediction_explainer` is set to `shap` for the SHAP prediction explainer.

2. Define values for each column to predict. The column names must match the feature column names in the trained table.

```
mysql> SET @variable = (JSON_OBJECT("column_name", value, "column_name", value, ...), model_handle, opt
```

In the following example, assign the data to analyze into the `@row_input` session variable.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 25,
          "workclass", "Private",
          "fnlwgt", 226802,
          "education", "11th",
          "education-num", 7,
          "marital-status", "Never-married",
          "occupation", "Machine-op-inspct",
          "relationship", "Own-child",
          "race", "Black",
          "sex", "Male",
          "capital-gain", 0,
          "capital-loss", 0,
          "hours-per-week", 40,
          "native-country", "United-States");
```

3. Run the `ML_EXPLAIN_ROW` routine.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);
```

In the following example run the same `shap` prediction explainer. Optionally, use `\G` to display the output in an easily readable format.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'shap')
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @census_model,
  JSON_OBJECT('prediction_explainer', 'shap')):
  {
    "age": 25,
    "sex": "Male",
    "race": "Black",
    "fnlwgt": 226802,
    "education": "11th",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "attributions": {
            "age_attribution": 0.03154012309521936,
            "sex_attribution": -0.002995059121088509,
            "race_attribution": 0.005126408998398765,
            "fnlwgt_attribution": -0.003139455788215409,
            "education_attribution": 0.0013752672453250653,
            "workclass_attribution": 0,
            "occupation_attribution": 0.020919219303459986,
            "capital-gain_attribution": 0.015089815859614985,
            "capital-loss_attribution": 0.0033537962775555263,
            "relationship_attribution": 0.027744370891787523,
            "education-num_attribution": 0.0284122832892542,
            "hours-per-week_attribution": 0.009110644648945954,
            "marital-status_attribution": 0.036222463769272406
        }
    },
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
```

```
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States",
    "age_attribution": 0.0315401231,
    "sex_attribution": -0.0029950591,
    "race_attribution": 0.005126409,
    "fnlwgt_attribution": -0.0031394558,
    "education_attribution": 0.0013752672,
    "workclass_attribution": 0,
    "occupation_attribution": 0.0209192193,
    "capital-gain_attribution": 0.0150898159,
    "capital-loss_attribution": 0.0033537963,
    "relationship_attribution": 0.0277443709,
    "education-num_attribution": 0.0284122833,
    "hours-per-week_attribution": 0.0091106446,
    "marital-status_attribution": 0.0362224638
}
1 row in set (4.3007 sec)
```

The output displays feature importance values for each column.

What's Next

- Review ML_EXPLAIN_ROW for parameter descriptions and options.
- Learn how to Generate Explanations for a Table.
- Learn how to Score a Model to get insight into the quality of the model.

4.5.6.2 Generating Prediction Explanations for a Table

`ML_EXPLAIN_TABLE` explains predictions for an entire table of unlabeled data. Explanations are performed in parallel.

Note

`ML_EXPLAIN_TABLE` is a very memory-intensive process. We recommend limiting the input table to a maximum of 100 rows. If the input table has more than ten columns, limit it to ten rows.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model

Unsupported Model Types

You cannot generate prediction explanations on a table for the following model types:

- Forecasting
- Recommendation
- Anomaly detection

- Anomaly detection for logs
- Topic modeling

Input Tables and Output Tables

You can specify the output table and the input table as the same table if all the following conditions are met:

- The input table does not have the columns that are created for the output table when generating predictions. Output columns are specific to each machine learning task. Some of these columns include:
  - `Prediction`
  - `ml_results`
  - `[input_column_name]_attribution`
- The input table does not have a primary key, and it does not have a column named `_4aad19ca6e_pk_id`. This is because `ML_EXPLAIN_TABLE` adds a column as the primary key with the name `_4aad19ca6e_pk_id` to the output table.

If you specify the output table and the input table as the same name, the predictions are inserted into the input table.

Preparing to Generate Explanations for a Table

Before running `ML_EXPLAIN_TABLE`, you must train, and then load the model you want to use.

1. The following example trains a dataset with the classification machine learning task.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'),
```

2. The following example loads the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, you can generate prediction explanations for a table. For parameter and option descriptions, see ML_EXPLAIN_TABLE.

Generating Explanations for a Table with the Default Permutation Importance Explainer

After training and loading a model, you can run `ML_EXPLAIN_TABLE` to generate a table of prediction explanations with the default Permutation Importance explainer>. However, if you train the `shap` prediction explainer with `ML_EXPLAIN`, you need to run `ML_EXPLAIN` again with the `permutation_importance` explainer before running `ML_EXPLAIN_TABLE` with the same explainer.

1. Run the `ML_EXPLAIN_TABLE` routine.

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
```

The following example runs `ML_EXPLAIN_TABLE` with the `permutation_importance` explainer.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_train_p
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `@census_model` is the session variable for the trained model.
- `census_data.census_train_permutation` is the fully qualified name of the output table that contains the explanations (*`schema_name.table_name`*).
- `prediction_explainer` is set to `permutation_importance` for the Permutation Importance prediction explainer.

2. Query the output table to review a sample of the results.

```
mysql> SELECT * FROM table_name LIMIT N;
```

The following example queries the top three rows of the output table.

```
mysql> SELECT * FROM census_train_permutation LIMIT 3;
+-------------------+-----+-----------+--------+------------+---------------+--------------------+---------
| _4aad19ca6e_pk_id | age | workclass | fnlwgt | education  | education-num | marital-status     | occupati
+-------------------+-----+-----------+--------+------------+---------------+--------------------+---------
|                 1 |  37 | Private   |  99146 | Bachelors  |            13 | Married-civ-spouse | Exec-mar
|                 2 |  34 | Private   |  27409 | 9th        |             5 | Married-civ-spouse | Craft-re
|                 3 |  30 | Private   | 299507 | Assoc-acdm |            12 | Separated          | Other-se
+-------------------+-----+-----------+--------+------------+---------------+--------------------+---------
```

The results display information on the columns that had the largest impact towards the predictions and the columns that contributed the most against the prediction.

A warning displays if the model is of low quality.

Generating Explanations for a Table with the SHAP Explainer

To generate a table of prediction explanations with the SHAP explainer, you must first run the SHAP explainer with `ML_EXPLAIN`.

1. Run the `ML_EXPLAIN` routine.

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name', model_handle, [options]);
```

The following example run the `shap` prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('census_data.census_train', 'revenue', @census_model, JSON_OBJECT('prediction_ex
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `revenue` is the name of the target column, which contains ground truth values.
- `@census_model` is the session variable for the trained model.
- `prediction_explainer` is set to `shap` for the SHAP prediction explainer.

2. Run the `ML_EXPLAIN_TABLE` routine.

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
```

The following example runs the `shap` prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_train_e
```

Where:

- `census_data.census_train` is the fully qualified name of the table that contains the training dataset (*`schema_name.table_name`*).
- `@census_model` is the session variable for the trained model.
- `census_data.census_train_explanations` is the fully qualified name of the output table that contains the explanations (*`schema_name.table_name`*).
- `prediction_explainer` is set to `shap` for the SHAP prediction explainer.

3. Query the output table to review a sample of the results.

```
mysql> SELECT * FROM table_name LIMIT N;
```

The following example queries the top three rows of the output table.

```
mysql> SELECT * FROM census_train_explanations LIMIT 3;
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-----
| _4aad19ca6e_pk_id | age | workclass | fnlwgt | education  | education-num | marital-status    | occu
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-----
|                 1 |  37 | Private   |  99146 | Bachelors  |           13 | Married-civ-spouse | Exec
|                 2 |  34 | Private   |  27409 | 9th        |            5 | Married-civ-spouse | Craf
|                 3 |  30 | Private   | 299507 | Assoc-acdm |           12 | Separated          | Othe
+-------------------+-----+-----------+--------+------------+--------------+-------------------+-----
```

The results display feature importance values for each column.

A warning displays if the model is of low quality.

What's Next

- Review ML_EXPLAIN_TABLE for parameter descriptions and options.
- Learn how to Score a Model to get insight into the quality of the model.

4.5.7 Scoring a Model

`ML_SCORE` scores a model by generating predictions using the feature columns in a labeled dataset as input and comparing the predictions to ground truth values in the target column of the labeled dataset.

You cannot score a model with a topic modeling task type.

Before You Begin

- Review the following:
  - Prepare Data
  - Train a Model
  - Load a Model
  - Generate Predictions
  - Generate Model Explanations

- Generate Prediction Explanations

ML_SCORE Overview

The dataset used with `ML_SCORE` should have the same feature columns as the dataset used to train the model, but the data sample should be different from the data used to train the model. For example, you might reserve 20 to 30 percent of a labeled dataset for scoring.

`ML_SCORE` returns a computed metric indicating the quality of the model. A value of `None` is reported if a score for the specified or default metric cannot be computed. If an invalid metric is specified, the following error message is reported: `Invalid data for the metric. Score could not be computed.`

Models with a low score can be expected to perform poorly, producing predictions and explanations that cannot be relied upon. A low score typically indicates that the provided feature columns are not a good predictor of the target values. In this case, consider adding more rows or more informative features to the training dataset.

You can also run `ML_SCORE` on the training dataset and a labeled test dataset and compare results to ensure that the test dataset is representative of the training dataset. A high score on a training dataset and low score on a test dataset indicates that the test data set is not representative of the training dataset. In this case, consider adding rows to the training dataset that better represent the test dataset.

AutoML supports a variety of scoring metrics to help you understand how your model performs across a series of benchmarks. The metric you select to score the model must be compatible with the `task` type and the target data. See Optimization and Scoring Metrics.

Preparing to Score a Model

Before running `ML_SCORE`, you must train, and then load the trained model you want to use for scoring.

1. The following example trains a dataset with the classification machine learning task.

   ```
   mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cen
   ```

2. The following example loads the trained model.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
   ```

For more information about training and loading models, see Train a Model and Load a Model.

After training and loading the model, prepare a table of labeled data to score that has a different set of data from the trained model. This is considered the validation dataset. For parameter and option descriptions, see ML_SCORE.

Scoring a Model

To score a model, run the `ML_SCORE` routine.

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

The following example uses the `accuracy` metric to compute model quality:

```
mysql> CALL sys.ML_SCORE('census_data.census_validate', 'revenue', @census_model, 'accuracy', @score, NULL);
```

Where:

- `census_data.census_validate` is the fully qualified name of the validation dataset table (*`schema_name.table_name`*).

- `revenue` is the name of the target column containing ground truth values.
- `@census_model` is the session variable that contains the model handle.
- `accuracy` is the scoring metric. For other supported scoring metrics, see Optimization and Scoring Metrics.
- `@score` is the user-defined session variable that stores the computed score. The `ML_SCORE` routine populates the variable. User variables are written as `@var_name`. The examples in this guide use `@score` as the variable name. Any valid name for a user-defined variable is permitted, for example `@my_score`.
- `NULL` sets no options for the routine. To view available options, see ML_SCORE.

To retrieve the computed score, query the `@score` session variable.

```
mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.8888888955116272 |
+--------------------+
1 row in set (0.0409 sec)
```

Review the score value and determine if the trained model is reliable enough for generating predictions and explanations.

What's Next

- Review ML_SCORE for parameter descriptions and options.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.6 Machine Learning Use Cases

4.6.1 Classify Data

Classification models predict the discrete value of input data to specific predefined categories. Some examples of classification include loan approvals, churn prediction, and spam detection.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The classification use-case is to approve or reject loan applications for clients based on their personal and socioeconomic status, assets, liabilities, credit rating, and past loan details.

To generate your own datasets for creating machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.1.1 Preparing Data for a Classification Model

This topic describes how to prepare the data to use for a classification machine learning model. It uses a data sample generated by OCI GenAI. The classification use-case is to approve or reject loan applications for clients based on their personal and socioeconomic status, assets, liabilities, credit rating, and past loan details. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 20 records, and the testing dataset has 10 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

Before You Begin

- Learn how to Prepare Data.

Preparing Data

To prepare the data for the classification model:

1. Connect to the MySQL Server.

2. Create and use the database to store the data.

```
mysql> CREATE DATABASE classification_data;
mysql> USE classification_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE Loan_Training (
    ClientID INT PRIMARY KEY,
    ClientAge INT NOT NULL,
    Gender VARCHAR(10) NOT NULL,
    Education VARCHAR(50) NOT NULL,
    Occupation VARCHAR(50) NOT NULL,
    Income REAL NOT NULL,
    Debt REAL NOT NULL,
    CreditScore INT NOT NULL,
    Assets REAL NOT NULL,
    Liabilities REAL NOT NULL,
    LoanType VARCHAR(20) NOT NULL,
    LoanAmount REAL NOT NULL,
    Approved VARCHAR(10) NOT NULL
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO Loan_Training (ClientID, ClientAge, Gender, Education, Occupation, Income, Debt, CreditScore, A
(101, 30, 'Male', 'Bachelor''s', 'Engineer', 75000, 15000, 700, 300000, 80000, 'Home', 250000, 'Approved'),
(102, 25, 'Female', 'Master''s', 'Analyst', 60000, 10000, 680, 200000, 50000, 'Personal', 120000, 'Rejected
(103, 40, 'Male', 'High School', 'Manager', 80000, 20000, 650, 450000, 120000, 'Business', 150000, 'Approve
(104, 35, 'Female', 'PhD', 'Doctor', 120000, 30000, 750, 600000, 250000, 'Car', 30000, 'Approved'),
(105, 28, 'Male', 'College', 'IT Specialist', 55000, 8000, 620, 280000, 90000, 'Education', 80000, 'Rejecte
(106, 45, 'Female', 'Bachelor''s', 'Teacher', 70000, 15000, 720, 500000, 180000, 'Home', 200000, 'Approved'
(107, 32, 'Male', 'Associate', 'Sales', 65000, 12000, 670, 350000, 100000, 'Vacation', 18000, 'Rejected'),
(108, 22, 'Female', 'College', 'Student', 30000, 5000, 660, 150000, 40000, 'Education', 10000, 'Approved'),
(109, 50, 'Male', 'Master''s', 'Lawyer', 110000, 40000, 780, 700000, 350000, 'Investment', 500000, 'Rejecte
(110, 38, 'Female', 'High School', 'Nurse', 52000, 18000, 640, 220000, 120000, 'Medical', 35000, 'Approved'
(111, 48, 'Male', 'Diploma', 'Plumber', 48000, 10000, 600, 180000, 70000, 'Home Improvement', 25000, 'Rejec
(112, 55, 'Female', 'Bachelor''s', 'Writer', 90000, 25000, 760, 400000, 200000, 'Retirement', 150000, 'Appr
(113, 36, 'Male', 'Master''s', 'Accountant', 78000, 22000, 740, 380000, 150000, 'Refinance', 200000, 'Appro
(114, 24, 'Female', 'College', 'Designer', 45000, 7000, 610, 250000, 100000, 'Startup', 50000, 'Rejected'),
(115, 42, 'Male', 'PhD', 'Scientist', 130000, 50000, 800, 550000, 300000, 'Research', 400000, 'Approved'),
(116, 52, 'Female', 'Master''s', 'Marketer', 85000, 35000, 770, 480000, 280000, 'Marketing', 120000, 'Rejec
(117, 34, 'Male', 'Bachelor''s', 'Programmer', 68000, 16000, 690, 320000, 110000, 'Equipment', 85000, 'Appr
(118, 26, 'Female', 'Associate', 'Retail', 42000, 6000, 630, 200000, 70000, 'Wedding', 28000, 'Rejected'),
(119, 46, 'Male', 'College', 'Pilot', 100000, 45000, 710, 520000, 250000, 'Boat', 350000, 'Approved'),
```

```
(120, 58, 'Female', 'PhD', 'Professor', 140000, 60000, 820, 650000, 450000, 'Real Estate', 550000, 'Rej
```

5. Create the table to use for generating predictions and explanations. This is the test dataset. It has the same columns as the training dataset, but the target column, `Approved`, is not considered when generating predictions or explanations.

```
mysql> CREATE TABLE Loan_Testing (
    ClientID INT PRIMARY KEY,
    ClientAge INT NOT NULL,
    Gender VARCHAR(10) NOT NULL,
    Education VARCHAR(50) NOT NULL,
    Occupation VARCHAR(50) NOT NULL,
    Income REAL NOT NULL,
    Debt REAL NOT NULL,
    CreditScore INT NOT NULL,
    Assets REAL NOT NULL,
    Liabilities REAL NOT NULL,
    LoanType VARCHAR(20) NOT NULL,
    LoanAmount REAL NOT NULL,
    Approved VARCHAR(10) NOT NULL
);
```

6. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO Loan_Testing (ClientID, ClientAge, Gender, Education, Occupation, Income, Debt, CreditScore
(201, 38, 'Male', 'College', 'Architect', 62000, 18000, 660, 380000, 160000, 'Home', 280000, 'Approved'
(202, 29, 'Female', 'Master''s', 'HR Manager', 58000, 12000, 690, 260000, 110000, 'Personal', 150000, '
(203, 44, 'Male', 'Bachelor''s', 'Chef', 72000, 25000, 730, 420000, 200000, 'Business', 180000, 'Approv
(204, 56, 'Female', 'PhD', 'Psychologist', 105000, 35000, 790, 580000, 320000, 'Car', 40000, 'Rejected'
(205, 31, 'Male', 'High School', 'Carpenter', 50000, 8000, 610, 240000, 85000, 'Education', 90000, 'App
(206, 27, 'Female', 'College', 'Artist', 48000, 7000, 640, 220000, 95000, 'Art', 150000, 'Rejected'),
(207, 49, 'Male', 'Associate', 'Electrician', 55000, 15000, 670, 300000, 120000, 'Home Improvement', 20
(208, 53, 'Female', 'Bachelor''s', 'Journalist', 88000, 30000, 750, 460000, 280000, 'Travel', 180000, '
(209, 37, 'Male', 'Master''s', 'Financial Advisor', 76000, 22000, 700, 360000, 150000, 'Investment', 25
(210, 23, 'Female', 'College', 'Intern', 35000, 5000, 600, 160000, 60000, 'Education', 20000, 'Rejected
```

What's Next

- Learn how to Train a Classification Model.

4.6.1.2 Training a Classification Model

After preparing the data for a classification model, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for a Classification Model.

Training the Model

Train the model with the `ML_TRAIN` routine and use the `training_data` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

```
mysql> SET @model='classification_use_case';
```

The model handle is set to `classification_use_case`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_handle
```

   Replace *table_name*, *target_column_name*, *task_name*, and *model_handle* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

```
mysql> CALL sys.ML_TRAIN('classification_data.Loan_Training', 'Approved', JSON_OBJECT('task', 'classificati
```

   Where:

   - `classification_data.Loan_Training` is the fully qualified name of the table that contains the training dataset (*database_name.table_name*).
   - `Approved` is the name of the target column, which contains ground truth values.
   - `JSON_OBJECT('task', 'classification')` specifies the machine learning task type.
   - `@model` is the session variable previously set that defines the model handle to the name defined by the user: `classification_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handl
+----------+--------------------------------------------+------------------------------------+
| model_id | model_handle                               | train_table_name                   |
+----------+--------------------------------------------+------------------------------------+
|        1 | classification_use_case                    | classification_data.Loan_Training  |
+----------+--------------------------------------------+------------------------------------+
```

What's Next

- Learn how to Generate Predictions for a Classification Model.

4.6.1.3 Generating Predictions for a Classification Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the `testing_data` dataset. Even though the table has labels for the `Approved` target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model
- Train a Classification Model

Generating Predictions for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('classification_data.Loan_Testing', @model, 'classification_data.Loan_
   ```

   Where:

   - `classification_data.Loan_Testing` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `classification_data.Loan_Testing_predictions` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
   - `NULL` sets no options for the routine.

3. Query the `Approved`, `Prediction`, and `ml_results` columns from the output table. This allows you to compare the real value with the generated prediction. You can also review the probabilities for each prediction. If needed, you can also query all the columns from the table (`SELECT * FROM classification_predictions`) to review all the data at once.

   ```
   mysql> SELECT Approved, Prediction, ml_results FROM Loan_Testing_predictions;
   +----------+------------+------------------------------------------------------------------------------
   | Approved | Prediction | ml_results
   +----------+------------+------------------------------------------------------------------------------
   | Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.983
   | Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.113
   | Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.986
   | Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.096
   | Approved | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.040
   | Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.108
   | Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.553
   | Rejected | Rejected   | {"predictions": {"Approved": "Rejected"}, "probabilities": {"Approved": 0.169
   | Approved | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.983
   | Rejected | Approved   | {"predictions": {"Approved": "Approved"}, "probabilities": {"Approved": 0.554
   +----------+------------+------------------------------------------------------------------------------
   10 rows in set (0.0430 sec)
   ```

The results show that two predictions do not match up with the real values.

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Learn now to Query Model Explanation and Generate Prediction Explanations for a Classification Model.

4.6.1.4 Query Model Explanation and Generate Prediction Explanations for a Classification Model

After training a classification model, you can query the default model explanation or query new model explanations. You can also generate prediction explanations. Explanations help you understand which features had the most influence on generating predictions.

Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model
- Train a Classification Model
- Generate Predictions for a Classification Model

Generating the Model Explanation

After training a model, you can query the default model explanation with the Permutation Importance explainer.

To generate explanations for other model explainers, see Generate Model Explanations and ML_EXPLAIN.

Query the `model_explanation` column from the model catalog and define the model handle previously created. Update `user1` with your own user name. Use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG
     WHERE model_handle='classification_use_case';
+-----------------------------------------------------------------------------------------------------+
| JSON_PRETTY(model_explanation)                                                                      |
+-----------------------------------------------------------------------------------------------------+
| {
  "permutation_importance": {
    "Debt": 0.5014,
    "Assets": 0.0,
    "Gender": 0.0,
    "Income": 0.0,
    "ClientID": 0.0,
    "LoanType": 0.0,
    "ClientAge": 0.1231,
    "Education": 0.0,
    "LoanAmount": 0.0,
```

```
    "Occupation": 0.0,
    "CreditScore": 0.0,
    "Liabilities": 0.0525
  }
} |
+----------------------------------------------------------------------------------------------------+
1 row in set (0.0382 sec)
```

Feature importance values display for each column.

Generating Prediction Explanations for a Table

After training a model, you can generate a table of prediction explanations on the `testing_data` dataset by using the default Permutation Importance prediction explainer.

To generate explanations for other model explainers, see Generate Prediction Explanations and ML_EXPLAIN_TABLE.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Use the `ML_EXPLAIN_TABLE` routine to generate explanations for predictions made in the test dataset.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_EXPLAIN_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE('classification_data.Loan_Testing', @model, 'classification_data.Loan_
           JSON_OBJECT('prediction_explainer', 'permutation_importance'));
   Query OK, 0 rows affected (12.2957 sec)
   ```

   Where:

   - `classification_data.Loan_Testing` is the fully qualified name of the test dataset.
   - `@model` is the session variable for the model handle.
   - `classification_data.Loan_Testing_explanations` is the fully qualified name of the output table with explanations.
   - `permutation_importance` is the selected prediction explainer to use to generate explanations.

3. Query `Notes` and `ml_results` from the output table to review which column contributed the most against or had the largest impact towards the prediction. You can also review individual attribution values for each column. Use `\G` to view the output in an easily readable format.

```
mysql> SELECT Notes, ml_results FROM Loan_Testing_explanations\G
*************************** 1. row ***************************
     Notes: Debt (18000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "Debt (18000.0) had the largest impact towar
*************************** 2. row ***************************
     Notes: ClientAge (29) had the largest impact towards predicting Rejected, whereas Debt (12000.0) contr
ml_results: {"attributions": {"Debt": -0.01, "Liabilities": 0.02, "ClientAge": 0.17, "LoanAmount": 0.08},
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (29) had the largest impact towar
*************************** 3. row ***************************
     Notes: Debt (25000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "Debt (25000.0) had the largest impact towar
*************************** 4. row ***************************
     Notes: ClientAge (56) had the largest impact towards predicting Rejected, whereas Debt (35000.0) contr
ml_results: {"attributions": {"Debt": -0.07, "Liabilities": 0.52, "ClientAge": 0.75, "LoanAmount": 0.01},
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (56) had the largest impact towar
*************************** 5. row ***************************
     Notes: LoanAmount (90000.0) had the largest impact towards predicting Rejected
ml_results: {"attributions": {"Debt": 0.0, "Liabilities": 0.01, "ClientAge": 0.1, "LoanAmount": 0.14},
             "predictions": {"Approved": "Rejected"}, "notes": "LoanAmount (90000.0) had the largest impact
*************************** 6. row ***************************
     Notes: ClientAge (27) had the largest impact towards predicting Rejected
ml_results: {"attributions": {"Debt": -0.0, "Liabilities": 0.01, "ClientAge": 0.16, "LoanAmount": 0.08},
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (27) had the largest impact towar
*************************** 7. row ***************************
     Notes: Debt (15000.0) had the largest impact towards predicting Approved, whereas ClientAge (49) contr
ml_results: {"attributions": {"Debt": 0.49, "Liabilities": -0.07, "ClientAge": -0.43, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "Debt (15000.0) had the largest impact towar
*************************** 8. row ***************************
     Notes: ClientAge (53) had the largest impact towards predicting Rejected, whereas Debt (30000.0) contr
ml_results: {"attributions": {"Debt": -0.13, "Liabilities": 0.56, "ClientAge": 0.68, "LoanAmount": -0.07},
             "predictions": {"Approved": "Rejected"}, "notes": "ClientAge (53) had the largest impact towar
*************************** 9. row ***************************
     Notes: Debt (22000.0) had the largest impact towards predicting Approved
ml_results: {"attributions": {"Debt": 0.87, "Liabilities": -0.0, "ClientAge": 0.0, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "Debt (22000.0) had the largest impact towar
*************************** 10. row ***************************
     Notes: No features had a significant impact on model prediction
ml_results: {"attributions": {"Debt": 0.0, "Liabilities": 0.0, "ClientAge": 0.0, "LoanAmount": 0.0},
             "predictions": {"Approved": "Approved"}, "notes": "No features had a significant impact on mod
10 rows in set (0.0461 sec)
```

To generate prediction explanations for one or more rows of data, see Generate Prediction Explanations for a Row of Data.

What's Next

- Learn how to Score a Classification Model.

4.6.1.5 Scoring a Classification Model

After generating predictions and explanations, you can score the model to assess its reliability. For a list of scoring metrics you can use with classification models, see Classification Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Complete the following tasks:

- Prepare Data for a Classification Model

- Train a Classification Model
- Generate Predictions for a Classification Model
- Query Model Explanation and Generate Prediction Explanations for a Classification Model

Scoring the Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('classification_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `accuracy` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('classification_data.Loan_Testing', 'Approved', @model, 'accuracy', @classific
   ```

   Where:

   - `classification_data.Loan_Testing` is the fully qualified name of the validation dataset.
   - `Approved` is the target column name with ground truth values.
   - `@model` is the session variable for the model handle.
   - `accuracy` is the selected scoring metric.
   - `@classification_score` is the session variable name for the score value.
   - `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @classification_score session variable.

   ```
   mysql> SELECT @classification_score;
   +----------------------+
   | @classification_score |
   +----------------------+
   |     0.800000011920929 |
   +----------------------+
   1 row in set (0.0431 sec)
   ```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('classification_use_case');
   ```

   To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

4.6.2 Perform Regression Analysis

Machine learning regression models generate predictions based on the relationship between a dependent variable and one or more independent variables. Some examples of regression analysis include predicting sales during different seasons, predicting purchasing behavior on a website based on the characteristics of website visitors, and predicting the sale price of residences based on their size.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The regression use-case is to predict house prices based on the size of the house, the address of the house, and the state the house is located in.

To generate your own datasets to create machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.2.1 Preparing Data for a Regression Model

This topic describes how to prepare the data to use for a regression machine learning model. It uses a data sample generated by OCI GenAI. The regression use-case is to predict house prices based on the size of the house, the address of the house, and the state the house is located in. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 20 records, and the testing dataset has 10 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

Before You Begin

- Learn how to Prepare Data.

Preparing Data

To prepare the data for the regression model:

1. Connect to the MySQL Server.

2. Create and use the database to store the data.

```
mysql> CREATE DATABASE regression_data;
mysql> USE regression_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE house_price_training (
    id INT PRIMARY KEY,
    house_size INT,
    address TEXT,
    state TEXT,
    price INT
```

```
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO house_price_training (id, house_size, address, state, price)
VALUES
    (1, 1500, '123 Main St', 'California', 500000),
    (2, 2000, '456 Elm St', 'Texas', 650000),
    (3, 1800, '789 Oak Ave', 'New York', 700000),
    (4, 1200, '222 Pine Rd', 'Florida', 420000),
    (5, 1600, '555 Maple Lane', 'Washington', 550000),
    (6, 2500, '888 River Blvd', 'California', 800000),
    (7, 1300, '333 Creek St', 'Texas', 480000),
    (8, 1700, '666 Mountain Rd', 'Colorado', 520000),
    (9, 1400, '999 Valley View', 'New York', 580000),
    (10, 1900, '111 Ocean Blvd', 'Florida', 620000),
    (11, 1550, '2222 Lake Dr', 'Illinois', 540000),
    (12, 2100, '3333 Forest Ave', 'Texas', 750000),
    (13, 1650, '4444 Desert Rd', 'Arizona', 570000),
    (14, 1250, '5555 Riverbank St', 'Washington', 450000),
    (15, 1850, '6666 Sky Blvd', 'California', 720000),
    (16, 1350, '7777 Meadow Lane', 'Ohio', 490000),
    (17, 2050, '8888 Hill St', 'New York', 850000),
    (18, 1450, '9999 Creek Rd', 'Florida', 590000),
    (19, 1750, '10101 Ocean Ave', 'Texas', 680000),
    (20, 1580, '11111 Pine St', 'Illinois', 560000);
```

5. Create the table to use for generating predictions and explanations. This is the test dataset. It has the same columns as the training dataset, but the target column, `price`, is not considered when generating predictions or explanations.

```
mysql> CREATE TABLE house_price_testing (
    id INT PRIMARY KEY,
    house_size INT,
    address TEXT,
    state TEXT,
    price INT
);
```

6. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO house_price_testing (id, house_size, address, state, price)
VALUES
    (1, 1400, '500 Elm St', 'Nevada', 470000),
    (2, 1900, '200 River Rd', 'California', 630000),
    (3, 1600, '300 Mountain Ave', 'Colorado', 530000),
    (4, 2200, '400 Lake Blvd', 'New York', 780000),
    (5, 1300, '500 Creek Lane', 'Texas', 460000),
    (6, 1700, '600 Valley View Rd', 'Florida', 510000),
    (7, 1500, '700 Ocean St', 'Washington', 500000),
    (8, 1800, '800 Sky Blvd', 'Oregon', 600000),
    (9, 1200, '900 Meadow Ave', 'Illinois', 430000),
    (10, 2100, '1000 Hill Rd', 'New Jersey', 760000);
```

What's Next

- Learn how to Train a Regression Model.

4.6.2.2 Training a Model for Regression

After preparing the data for a regression model, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for a Regression Model.

Training the Model

Train the model with the ML_TRAIN routine and use the house_price_training table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

   Replace *@variable* and *model_handle* with your own definitions. For example:

```
mysql> SET @model='regression_use_case';
```

   The model handle is set to regression_use_case.

2. Run the ML_TRAIN routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @variable);
```

   Replace *table_name*, *target_column_name*, *task_name*, and *variable* with your own values.

   The following example runs ML_TRAIN on the training dataset previously created.

```
mysql> CALL sys.ML_TRAIN('regression_data.house_price_training', 'price', JSON_OBJECT('task', 'regression')
```

   Where:

   - regression_data.house_price_training is the fully qualified name of the table that contains the training dataset (*database_name.table_name*).
   - price is the name of the target column, which contains ground truth values.
   - JSON_OBJECT('task', 'regression') specifies the machine learning task type.
   - @model is the session variable previously set that defines the model handle to the name defined by the user: regression_use_case. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the @model session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handl
+----------+-----------------------------------------------+---------------------------------------+
| model_id | model_handle                                  | train_table_name                      |
+----------+-----------------------------------------------+---------------------------------------+
|        2 | regression_use_case                           | regression_data.house_price_training  |
+----------+-----------------------------------------------+---------------------------------------+
```

What's Next

- Learn how to Generate Predictions for a Regression Model.

4.6.2.3 Generating Predictions for a Regression Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the house_price_testing dataset. Even though the table has labels for the price target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model
- Train a Regression Model

Generating Predictions for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to NULL.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the ML_PREDICT_TABLE routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *table_name*, *model_handle*, and *output_table_name* with your own values. Add *options* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs ML_PREDICT_TABLE on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('regression_data.house_price_testing', @model, 'regression_data.house_
   ```

   Where:

   - regression_data.house_price_testing is the fully qualified name of the input table that contains the data to generate predictions for (*database_name.table_name*).
   - @model is the session variable for the model handle.
   - regression_data.house_price_predictions is the fully qualified name of the output table with predictions (*database_name.table_name*).
   - NULL sets no options for the routine.

3. Query the price, Prediction, and ml_results columns from the output table. This allows you to compare the real value with the generated prediction. If needed, you can also query all the columns from the table (SELECT * FROM house_price_predictions) to review all the data at once.

   ```
   mysql> SELECT price, Prediction, ml_results FROM house_price_predictions;
   ```

```
+--------+------------+----------------------------------------+
| price  | Prediction | ml_results                             |
+--------+------------+----------------------------------------+
| 470000 |     534372 | {"predictions": {"price": 534371.5625}}  |
| 630000 |     669040 | {"predictions": {"price": 669040.125}}   |
| 530000 |     512676 | {"predictions": {"price": 512676.40625}} |
| 780000 |     794059 | {"predictions": {"price": 794059.0}}     |
| 460000 |     489206 | {"predictions": {"price": 489206.0}}     |
| 510000 |     534240 | {"predictions": {"price": 534239.8125}}  |
| 500000 |     532544 | {"predictions": {"price": 532543.9375}}  |
| 600000 |     698540 | {"predictions": {"price": 698539.9375}}  |
| 430000 |     454276 | {"predictions": {"price": 454275.5}}     |
| 760000 |     794059 | {"predictions": {"price": 794059.0}}     |
+--------+------------+----------------------------------------+
10 rows in set (0.0417 sec)
```

Review the predictions and compare with the real prices.

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Learn now to Query Model Explanation and Generate Prediction Explanations for a Regression Model.

4.6.2.4 Query Model Explanation and Generate Prediction Explanations for a Regression Model

After training a regression model, you can query the default model explanation or query new model explanations. You can also generate prediction explanations. Explanations help you understand which features had the most influence on generating predictions.

Feature importance is presented as an attribution value ranging from -1 to 1. A positive value indicates that a feature contributed toward the prediction. A negative value indicates that the feature contributes positively towards one of the other possible predictions.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model
- Train a Regression Model
- Generate Predictions for a Regression Model

Generating the Model Explanation

After training a model, you can query the default model explanation with the Permutation Importance explainer.

To generate explanations for other model explainers, see Generate Model Explanations and ML_EXPLAIN.

Query the `model_explanation` column from the model catalog and define the model handle previously created. Update `user1` with your own user name. Use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG
                     WHERE model_handle='regression_use_case';
```

```
+------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "id": 0.0257,
    "state": 0.0278,
    "address": 0.0,
    "house_size": 2.3762
  }
} |
+------------------------------------------------------------------------------------------
1 row in set (0.000 sec)
```

Feature importance values display for each column.

Generating Prediction Explanations for a Table

After training a model, you can generate a table of prediction explanations on the `house_price_testing` dataset by using the default Permutation Importance prediction explainer.

To generate explanations for other model explainers, see Generate Prediction Explanations and ML_EXPLAIN_TABLE.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Use the `ML_EXPLAIN_TABLE` routine to generate explanations for predictions made in the test dataset.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_EXPLAIN_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_EXPLAIN_TABLE('regression_data.house_price_testing', 'regression_use_case', 'regress
           JSON_OBJECT('prediction_explainer', 'permutation_importance'));
   ```

   Where:

   - `regression_data.house_price_testing` is the fully qualified name of the test dataset.
   - `regression_use_case` is the model handle for the trained table.
   - `regression_data.regression_explanations` is the fully qualified name of the output table with explanations.
   - `permutation_importance` is the selected prediction explainer to use to generate explanations.

3. Query `Notes` and `ml_results` from the output table to review which column contributed the most against or had the largest impact towards the prediction. You can also review individual attribution values for each column. Use `\G` to view the output in an easily readable format.

```
mysql> SELECT Notes, ml_results FROM regression_data.regression_explanations\G
*************************** 1. row ***************************
     Notes: house_size (1400) increased the value the model predicted the most, whereas state (Nevada) redu
ml_results: {"attributions": {"house_size": 101328.28, "state": -1037.94, "id": -300.23}, "predictions": {"
*************************** 2. row ***************************
     Notes: house_size (1900) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 235996.83, "state": 16140.48, "id": 0.06}, "predictions": {"pri
*************************** 3. row ***************************
     Notes: house_size (1600) increased the value the model predicted the most, whereas state (Colorado) re
ml_results: {"attributions": {"house_size": 79633.12, "state": -1220.23, "id": 5602.78}, "predictions": {"p
*************************** 4. row ***************************
     Notes: house_size (2200) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 361015.72, "state": 9903.62, "id": 12578.75}, "predictions": {"
*************************** 5. row ***************************
     Notes: house_size (1300) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 31384.31, "state": 226.31, "id": 30184.16}, "predictions": {"pr
*************************** 6. row ***************************
     Notes: house_size (1700) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 80747.0, "state": 7330.35, "id": 24427.78}, "predictions": {"pr
*************************** 7. row ***************************
     Notes: house_size (1500) increased the value the model predicted the most, whereas state (Washington)
ml_results: {"attributions": {"house_size": 79051.12, "state": -1316.08, "id": 28659.66}, "predictions": {"
*************************** 8. row ***************************
     Notes: house_size (1800) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 245256.83, "state": 8604.06, "id": 12578.75}, "predictions": {"
*************************** 9. row ***************************
     Notes: id (9) increased the value the model predicted the most, whereas state (Illinois) reduced the v
ml_results: {"attributions": {"house_size": -0.03, "state": -0.03, "id": 21232.22}, "predictions": {"price"
*************************** 10. row ***************************
     Notes: house_size (2100) increased the value the model predicted the most
ml_results: {"attributions": {"house_size": 339783.47, "state": 10981.75, "id": 12411.04}, "predictions": {
```

To generate prediction explanations for one or more rows of data, see Generate Prediction Explanations for a Row of Data.

What's Next

- Learn how to Score a Regression Model.

4.6.2.5 Scoring a Regression Model

After generating predictions and explanations, you can score the model to assess its reliability. For a list of scoring metrics you can use with regression models, see Regression Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Complete the following tasks:

- Prepare Data for a Regression Model
- Train a Regression Model
- Generate Predictions for a Regression Model
- Query Model Explanation and Generate Prediction Explanations for a Regression Model

Scoring the Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('regression_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `r2` metric.

   ```
   mysql> CALL sys.ML_SCORE('regression_data.house_price_testing', 'price', 'regression_use_case', 'r2', @
   ```

   Where:

   - `regression_data.house_price_testing` is the fully qualified name of the validation dataset.
   - `price` is the target column name with ground truth values.
   - `'regression_use_case'` is the model handle for the trained model.
   - `r2` is the selected scoring metric.
   - `@regression_score` is the session variable name for the score value.
   - `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @regression_score session variable.

   ```
   mysql> SELECT @regression_score;
   +--------------------+
   | @regression_score  |
   +--------------------+
   | 0.8524690866470337 |
   +--------------------+
   1 row in set (0.0453 sec)
   ```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('regression_use_case');
   ```

   To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

4.6.3 Generating Forecasts

Forecasting models generate predictions on timeseries data. Some examples of forecasting include predicting the closing price of a stock, predicting the number of units sold in a day, and predicting the average price of gasoline.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The forecasting use-case is a univariate forecasting model that captures the monthly demand for electricity in San Francisco, California.

To generate your own datasets to create machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.3.1 Forecasting Task Types

This topic describes the types of forecasting models supported by AutoML.

Before You Begin

- Review the list of supported Forecasting Models.

You can create the following types of forecasting models.

Univariate Models

In a univariate model, you define one numeric column as an endogenous variable, specified as a `JSON_ARRAY`. This is the target column that AutoML forecasts. For example, you forecast the rainfall for the next month by using the past daily rainfall as an endogenous variable.

Multivariate Models

In a multivariate model, you define multiple columns as endogenous variables, specified as a `JSON_ARRAY`. You must define one of these columns as the target column (the column with ground truth values). For example, you forecast the rainfall for the next month by using the past rainfall, temperature highs and lows, atmospheric pressure, and humidity. The target column is rainfall.

Univariate and Multivariate Models with Exogenous Variables

You have the option to define exogenous variables for univariate and multivariate models. These columns have independent, non-forecast, predictive variables. For example, you forecast future sales and use weather conditions like rainfall and high and low daily temperature values as exogenous variables.

Selecting Forecasting Models

To specify which models that are considered for training, use the `model_list` option and enter the appropriate model names. If only one model is set for `model_list`, then only that model is considered. Review the list of supported Forecasting Models and which type of model they support, univariate endogenous models, univariate endogenous models with exogenous variables, and multivariate endogenous models with exogenous variables. .

If the `model_list` option is not set, then `ML_TRAIN` considers all supported models during the algorithm selection stage. If *`options`* includes `exogenous_variables`, all supported models are still considered, including models that do not support `exogenous_variables`.

For example, if *`options`* includes univariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` considers `NaiveForecaster`, `ThetaForecaster`, `ExpSmoothForecaster`, ETSForecaster, STLwESForecaster, STLwARIMAForecaster, SARIMAXForecaster, and OrbitForecaster. ML_TRAIN ignores exogenous_variables if the model does not support them.

Similarly, if *options* includes multivariate endogenous_variables with exogenous_variables, then ML_TRAIN considers VARMAXForecaster and DynFactorForecaster.

If *options* also includes include_column_list, this forces ML_TRAIN to only consider those models that support exogenous_variables.

What's Next

- Learn more about Prediction Intervals.
- Learn how to Train a Forecasting Model.

4.6.3.2 Prediction Intervals

Prediction intervals for forecasting models specify upper and lower bounds on predictions for forecasting based on level of confidence. For example, for a prediction interval of 0.95 with a lower bound of 25 units and an upper bound of 65 units, you are 95% confident that product ABC will sell between 25 and 65 units on a randomly selected day.

The prediction_interval option is included for the ML_PREDICT_TABLE routine, which specifies a level of confidence. Predictions provide three outputs corresponding to each endogenous variable: the forecasted value, a lower bound, and an upper bound.

For the prediction_interval option:

- The default value is 0.95.
- The data type for this value must be FLOAT.
- The value must be greater than 0 and less than 1.0.

What's Next

- Learn how to Train a Forecasting Model.

4.6.3.3 Preparing Data for a Forecasting Model

This topic describes how to prepare the data to use for a forecasting machine learning model. It uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 37 records, and the testing dataset has 4 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

Before You Begin

- Learn how to Prepare Data.

Preparing Data

To prepare the data for the forecasting model:

1. Connect to the MySQL Server.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE forecasting_data;
mysql> USE forecasting_data;
```

3. Create the table that is the sample dataset.

```
mysql> CREATE TABLE electricity_demand (
    date DATE PRIMARY KEY,
    demand FLOAT NOT NULL,
    temperature FLOAT NOT NULL
);
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO electricity_demand (date, demand, temperature) VALUES
('2022-01-01', 929.00, 53.53),
('2022-01-31', 949.69, 60.80),
('2022-03-02', 1160.84, 69.28),
('2022-04-01', 1054.52, 74.48),
('2022-05-01', 1061.40, 71.06),
('2022-05-31', 1012.36, 58.05),
('2022-06-30', 1098.87, 51.90),
('2022-07-30', 964.31, 39.70),
('2022-08-29', 1026.06, 32.47),
('2022-09-28', 995.23, 30.82),
('2022-10-28', 1076.04, 32.97),
('2022-11-27', 1059.46, 42.91),
('2022-12-27', 1060.97, 51.52),
('2023-01-26', 1153.59, 60.24),
('2023-02-25', 1204.72, 68.21),
('2023-03-27', 1203.33, 70.67),
('2023-04-26', 1218.42, 70.31),
('2023-05-26', 1163.28, 59.59),
('2023-06-25', 1161.86, 50.63),
('2023-07-25', 1131.38, 38.29),
('2023-08-24', 1138.72, 27.57),
('2023-09-23', 1119.34, 31.31),
('2023-10-23', 1090.38, 34.41),
('2023-11-22', 1213.87, 38.52),
('2023-12-22', 1219.91, 54.54),
('2024-01-21', 1193.49, 57.09),
('2024-02-20', 1326.44, 67.41),
('2024-03-21', 1274.64, 69.63),
('2024-04-20', 1325.90, 70.39),
('2024-05-20', 1351.45, 62.94),
('2024-06-19', 1306.45, 50.31),
('2024-07-19', 1341.97, 40.76),
('2024-08-18', 1214.96, 30.90),
('2024-09-17', 1300.12, 26.04),
('2024-10-17', 1262.46, 31.98),
('2024-11-16', 1281.46, 40.31),
('2024-12-16', 1331.06, 52.46),
('2025-01-15', 1379.42, 62.40),
('2025-02-14', 1426.11, 66.55),
('2025-03-16', 1381.74, 69.40),
('2025-04-15', 1488.34, 65.22);
```

5. Create the table to use as the training dataset. It retrieves some of the data from the sample dataset.

```
mysql> CREATE TABLE electricity_demand_train AS SELECT * FROM electricity_demand WHERE date < '2025-01-01';
```

6. Create the table to use for generating predictions. This is the test dataset. It retrieves the data from the sample dataset not used for the training dataset. It has the same columns as the training dataset, but the target column, `demand`, is not considered when generating predictions.

```
mysql> CREATE TABLE electricity_demand_test AS SELECT * FROM electricity_demand WHERE date >= '2025-01-01';
```

What's Next

- Learn how to Train a Forecasting Model.

4.6.3.4 Training a Forecasting Model

After preparing the data for a forecasting model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Forecasting Training
- Forecasting Options
- Unsupported Routines
- Training the Model
- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for a Forecasting Model.

Requirements for Forecasting Training

Define the following required parameters to train a forecasting model.

- Set the `task` parameter to `forecasting`.
- `datetime_index`: Define the column that has date and time data. The model uses this column as an index for the forecast variable. The following data types for this column are supported: `DATETIME`, `TIMESTAMP`, `DATE`, `TIME`, and `YEAR`, or an auto-incrementing index.

  The forecast models `SARIMAXForecaster`, `VARMAXForecaster`, and `DynFactorForecaster` cannot back test, that is forecast into training data, when using `exogenous_variables`. Therefore, the predict table must not overlap the `datetime_index` with the training table. The start date in the predict table must be a date immediately following the last date in the training table when `exogenous_variables` are used. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023. The predict table cannot start with year, for example, 2025 or 2030, because that would miss out 1 and 6 years, respectively.

  When *`options`* do not include `exogenous_variables` , the predict table can overlap the `datetime_index` with the training table. This supports back testing, with the exception of the following models: SARIMAXForecaster, VARMAXForecaster, and DynFactorForecaster.

  The valid range of years for `datetime_index` dates must be between 1678 and 2261. An error is returned if any part of the training table or predict table has dates outside this range. The last date in the training table plus the predict table length must still be inside the valid year range. For example, if the `datetime_index` in the training table has `YEAR` data type, and the last date is year 2023, the predict table length must be less than 238 rows: 2261 minus 2023 equals 238 rows.
- `endogenous_variables`: Define the column or columns to be forecast. One of these columns must also be specified as the `target_column_name`.

Forecasting Options

Based on the type of forecasting model you train, set the appropriate `JSON` options:

- `exogenous_variables`: Define the column or columns that have independent, non-forecast, predictive variables. These optional variables are not forecast, but help to predict the future values of the forecast variables. These variables affect a model without being affected by it. For example, for sales forecasting these variables might be advertising expenditure, occurrence of promotional events, weather, or holidays. Review Forecasting Models to see which models support exogenous variables.
- `model_list`: Set the type of forecasting model algorithm. See Forecasting Models to review supported algorithms.
- `include_column_list`: Define the columns of `exogenous_variables` that must be included for training and should not be dropped.

Unsupported Routines

You cannot run the following routines for a trained forecasting model:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`
- `ML_PREDICT_ROW`

Training the Model

After following the steps to Prepare Data for Forecasting Model, train the model with the `ML_TRAIN` routine and use the `electricity_demand_training` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @model='forecasting_use_case';
   ```

   The model handle is set to `forecasting_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_handle
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('forecasting_data.electricity_demand_train', 'demand',
          JSON_OBJECT('task', 'forecasting', 'datetime_index', 'date', 'endogenous_variables', JSON_ARRAY('dem
   ```

   Where:

   - `forecasting_data.electricity_demand_train` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).

- `demand` is the name of the target column, which contains ground truth values.
- The `JSON_OBJECT` defines the following:
  - `'task', 'forecasting'` specifies the machine learning task type.
  - `'datetime_index', 'date'` defines the `date` column as the one with data and time data.
  - `'endogenous_variables', JSON_ARRAY('demand')` defines the endogenous variables in a `JSON_ARRAY`. Since it is a univariate model, the only endogenous variable is `demand`.
- `@model` is the session variable previously set that defines the model handle to the name defined by the user: `forecasting_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_h
+----------+----------------------------------------------+-----------------------------------------
| model_id | model_handle                                 | train_table_name
+----------+----------------------------------------------+-----------------------------------------
|        3 | forecasting_use_case                         | forecasting_data.electricity_demand_train
+----------+----------------------------------------------+-----------------------------------------
```

What's Next

- Learn how to Generate Predictions for a Forecasting Model.
- Review additional Syntax Examples for Forecast Training

4.6.3.5 Generating Predictions for a Forecasting Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the `electricity_demand_test` dataset. Even though the table has labels for the `demand` target column, the column is not considered when generating predictions. This allows you to compare the predictions to the actual values in the dataset and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using unlabeled datasets for generating predictions.

The `datetime_index` column must be included. If using `exogenous_variables`, they must also be included. Any extra columns, for example `endogenous_variables`, are ignored for the prediction, but included in the output table.

Prediction interval values are included in the prediction results. See Prediction Intervals to learn more.

You cannot run `ML_PREDICT_ROW` with forecasting models.

Before You Begin

Complete the following tasks:

- Prepare Data for a Forecasting Model.

- Review how to Train a Forecasting Model.

Generating Forecasts for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('forecasting_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('forecasting_data.electricity_demand_test', @model, 'forecasting_data.elec
   ```

   Where:

   - `forecasting_data.electricity_demand_test` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).

   - `@model` is the session variable for the model handle.

   - `forecasting_data.electricity_demand_predictions` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).

   - `NULL` sets no options for the routine.

3. Query the `demand`, and `ml_results` columns from the output table. This allows you to compare the real demand with the generated forecast. You can also review the lower bound and upper bound prediction interval values for each forecast. Since no prediction interval value is set when running `ML_PREDICT_TABLE`, the default value of 0.95 is used.

   ```
   mysql> SELECT demand, ml_results FROM electricity_demand_predictions;
   +---------+------------------------------------------------------------------------------------------------
   | demand  | ml_results
   +---------+------------------------------------------------------------------------------------------------
   | 1379.42 | {"predictions": {"demand": 1316.5263873105694, "prediction_interval_demand": [1312.648750452689
   | 1426.11 | {"predictions": {"demand": 1322.148597544633, "prediction_interval_demand": [1317.7966015800637
   | 1381.74 | {"predictions": {"demand": 1327.6276527841787, "prediction_interval_demand": [1322.848069997051
   | 1488.34 | {"predictions": {"demand": 1332.9671980996688, "prediction_interval_demand": [1327.795189107038
   +---------+------------------------------------------------------------------------------------------------
   ```

What's Next

- Learn how to Score a Forecasting Model

4.6.3.6 Scoring a Forecasting Model

After generating predictions, you can score the model to assess its reliability. For a list of scoring metrics you can use with forecasting models, see Forecasting Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

The `ML_SCORE` routine does not require a `target_column_name` for forecasting, so you can set it to `NULL`. However, the target column needs to be in the table to generate a valid score value.

Before You Begin

Complete the following tasks:

- Prepare Data for a Forecasting Model
- Train a Forecasting Model
- Generate Predictions for a Forecasting Model

Scoring the Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('forecasting_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `neg_sym_mean_abs_percent_error` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('forecasting_data.electricity_demand_test', 'demand', @model, 'neg_sym_mean_ab
   ```

   Where:

   - `forecasting_data.electricity_demand_test` is the fully qualified name of the validation dataset.
   - `demand` is the target column name with ground truth values.
   - `@model` is the session variable for the model handle.
   - `neg_sym_mean_abs_percent_error` is the selected scoring metric.
   - `@forecasting_score` is the session variable name for the score value.

- NULL means that no other options are defined for the routine.

3. Retrieve the score by querying the @forecasting_score session variable.

```
mysql> SELECT @forecasting_score;
+----------------------+
| @forecasting_score   |
+----------------------+
| -0.06810028851032257 |
+----------------------+
```

4. If done working with the model, unload it with the ML_MODEL_UNLOAD routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('forecasting_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

4.6.4 Detect Anomalies

Anomaly detection, which is also known as outlier detection, is the machine learning task that finds unusual patterns in data.

AutoML supports unsupervised and semi-supervised anomaly detection. See Anomaly Detection Learning Types to learn more.

The following tasks use datasets generated by OCI GenAI using Meta Llama Models. The anomaly detection use-cases are to find unusual patterns of purchasing behavior for credit card transactions, and to find anomalies in log data.

To generate your own datasets to create machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.4.1 Anomaly Detection Model Types

You can use the following anomaly detection model types:

- GkNN (Generalized kth Nearest Neighbors)
- PCA (Principal Component Analysis)
- GLOF (Generalized Local Outlier Factor)

GkNN Model

Generalized kth Nearest Neighbors (GkNN) is an algorithm model developed at Oracle. It is a single ensemble algorithm that outperforms state-of-the-art models on public benchmarks. It can identify common anomaly types, such as local, global, and clustered anomalies, and can achieve an AUC score that is similar to, or better than, when identifying the following:

- Global anomalies compared to KNN, with an optimal k hyperparameter value.
- Local anomalies compared to LOF, with an optimal k hyperparameter value.
- Clustered anomalies.

Optimal k hyperparameter values would be extremely difficult to set without labels and knowledge of the use-case.

Other algorithms would require training and comparing scores from at least three algorithms to address global and local anomalies, ignoring clustered anomalies: LOF for local, KNN for global, and another generic method to establish a 2/3 voting mechanism.

What's Next

- Learn more about the following:
  - Anomaly Detection Learning Types
  - Anomaly Detection for Logs

4.6.4.2 Anomaly Detection Learning Types

The AutoML feature of MySQL AI provides two types of learning for anomaly detection models: unsupervised and semi-supervised.

Unsupervised Anomaly Detection

When running an unsupervised anomaly detection model, the machine learning algorithm requires no labeled data. When training the model, the `target_column_name` parameter must be set to `NULL`.

Semi-supervised Anomaly Detection

Semi-supervised learning for anomaly detection uses a specific set of labeled data along with unlabeled data to detect anomalies. To enable this, use the `experimental` and `semisupervised` options. The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component.

What's Next

- Learn more about the following:
  - Anomaly Detection Algorithm Model Types
  - Anomaly Detection for Logs
- Learn how to Prepare Data for an Anomaly Detection Model.

4.6.4.3 Anomaly Detection for Logs

Anomaly detection for logs allows you to detect anomalies in log data. To perform anomaly detection on logs, log data is cleaned, segmented, and encoded before running anomaly detection. This feature leverages the log template miner Drain3.

Consider the following when running anomaly detection on logs.

- The input table can only have the following columns:

    - The column containing the logs.
    - If including logs from different sources, a column containing the source of each log. The values in this column contain the names of the sources that each log belongs to. These values are used to group each host's logs together. If this column is not present, it is assumed that all logs originate from the same source.
    - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
    - At least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
- If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude irrelevant columns.
- The data collected for anomaly detection can be unsupervised or semi-supervised. To run semi-supervised anomaly detection, you can provide a separate column in the input table with labels for the labeled log lines. This column labels identified anomalous logs with a value of 1, non-anomalous logs with 0, and unlabeled logs with NULL. See Semi-supervised Anomaly Detection to learn more.
- In addition to the anomaly scores included in the output table, you have the option to leverage the GenAI feature of MySQL AI to provide textual log summaries.
- By default the following parameters are masked in the input data (training or test data): IP, DATETIME, TIME, HEX, IPPORT, and OCID. You have the option to mask additional regex patterns with the `additional_masking_regex` option.

What's Next

- Learn more about the following:
    - Anomaly Detection Algorithm Model Types
    - Anomaly Detection Learning Types
- Learn how to Prepare Data for an Anomaly Detection Model.

4.6.4.4 Preparing Data for an Anomaly Detection Model

This topic describes how to prepare the data to use for two anomaly detection machine learning models: a semi-supervised anomaly detection model, and an unsupervised anomaly detection model for logs. It uses data samples generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. In a real-life use case, you should prepare a larger amount of records than these data samples for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

This topic has the following sections.

- Before You Begin
- Preparing Data for a Semi-Supervised Anomaly Detection Model

- Preparing Data for an Unsupervised Anomaly Detection Model for Logs
- What's Next

Before You Begin

- Learn how to Prepare Data.

Preparing Data for a Semi-Supervised Anomaly Detection Model

The semi-supervised anomaly detection model looks for unusual patterns in credit card transactions. The data has a column, `target`, that has three possible values: 0 for normal, 1 for anomalous, and NULL for unlabeled.

To prepare the data for the semi-supervised anomaly detection model:

1. Connect to the MySQL Server.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE anomaly_data;
mysql> USE anomaly_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE credit_card_train (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    home_address VARCHAR(100),
    purchase_location VARCHAR(100),
    purchase_amount DECIMAL(10, 2),
    purchase_time DATETIME,
    target INT
);
```

4. Insert the sample data to train into the table. Copy and paste the following commands.

```
INSERT INTO credit_card_train (home_address, purchase_location, purchase_amount, purchase_time, target)
VALUES
    ('123 Main St, City A', 'Store X, City A', 50.75, '2023-08-01 14:30:00', 0),
    ('456 Elm St, City B', 'Cafe B, City B', 15.20, '2023-08-02 09:45:00', 1),
    ('789 Oak Ave, City C', 'Online Shop', 250.00, '2023-08-03 18:10:00', 0),
    ('222 Maple Lane, City A', 'Grocery Store A', 35.50, '2023-08-04 11:00:00', NULL),
    ('555 River Rd, City D', 'Electronics Store, City D', 800.50, '2023-08-05 16:20:00', 1),
    ('1010 Mountain View, City E', 'Boutique, City E', 120.30, '2023-08-06 10:35:00', 0),
    ('333 Ocean Blvd, City F', 'Convenience Store, City F', 20.15, '2023-08-07 19:50:00', NULL),
    ('666 Sky St, City G', 'Luxury Store, City G', 1500.00, '2023-08-08 12:00:00', 1),
    ('999 Green Valley, City H', 'Hardware Store, City H', 75.90, '2023-08-09 08:40:00', 0),
    ('111 Sunset Ave, City A', 'Store X, City A', 60.40, '2023-08-10 15:10:00', NULL),
    ('2222 Country Road, City B', 'Cafe B, City B', 28.75, '2023-08-11 07:30:00', 0),
    ('3333 Lakeside, City C', 'Online Shop', 180.25, '2023-08-12 13:20:00', 1),
    ('4444 Forest Glade, City D', 'Grocery Store, City D', 45.60, '2023-08-13 09:50:00', 0),
    ('5555 Meadow Lane, City E', 'Electronics Store, City E', 300.75, '2023-08-14 17:40:00', NULL),
    ('6666 Creekside, City F', 'Boutique, City F', 95.50, '2023-08-15 11:30:00', 1),
    ('7777 Hillcrest, City G', 'Convenience Store, City G', 12.80, '2023-08-16 18:50:00', 0),
    ('8888 Riverbank, City H', 'Luxury Store, City H', 2200.00, '2023-08-17 14:10:00', NULL),
    ('9999 Sunrise Blvd, City A', 'Hardware Store, City A', 55.25, '2023-08-18 09:30:00', 0),
    ('101010 Ocean View, City B', 'Store X, City B', 70.50, '2023-08-19 16:40:00', 1),
    ('111111 Mountain Rd, City C', 'Cafe C, City C', 32.90, '2023-08-20 11:20:00', NULL),
    ('121212 Downtown, City D', 'Online Shop', 450.00, '2023-08-21 17:50:00', 0),
    ('131313 Lakeside Ave, City E', 'Grocery Store, City E', 28.50, '2023-08-22 10:10:00', 1),
    ('141414 Green Park, City F', 'Electronics Store, City F', 650.75, '2023-08-23 15:30:00', 0),
    ('151515 Skyway, City G', 'Boutique, City G', 180.40, '2023-08-24 08:50:00', NULL),
    ('161616 Meadow View, City H', 'Convenience Store, City H', 35.10, '2023-08-25 13:40:00', 0),
    ('171717 River Rd, City A', 'Luxury Store, City A', 1300.50, '2023-08-26 19:20:00', 1),
```

```
    ('181818 Sunset Blvd, City B', 'Hardware Store, City B', 85.60, '2023-08-27 12:30:00', NULL),
    ('191919 Country Lane, City C', 'Store Y, City C', 150.20, '2023-08-28 07:40:00', 0),
    ('202020 Forest Edge, City D', 'Cafe D, City D', 42.75, '2023-08-29 14:50:00', 1),
    ('212121 Lakeside View, City E', 'Online Shop', 220.50, '2023-08-30 09:20:00', 0),
    ('222222 Creekside Ave, City F', 'Grocery Store, City F', 55.90, '2023-08-31 16:10:00', NULL);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset. The target column, `target`, is used for the sem-supervised component of the training.

```
mysql> CREATE TABLE credit_card_test (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    home_address VARCHAR(100),
    purchase_location VARCHAR(100),
    purchase_amount DECIMAL(10, 2),
    purchase_time DATETIME,
    target INT
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO credit_card_test (home_address, purchase_location, purchase_amount, purchase_time, target)
VALUES
    ('3030 Riverbank Dr, City I', 'Grocery Store, City I', 52.30, '2023-09-01 10:30:00', 0),
    ('3131 Mountain Rd, City J', 'Electronics Store, City J', 120.50, '2023-09-02 16:45:00', 0),
    ('3232 Ocean Ave, City K', 'Boutique, City K', 85.20, '2023-09-03 11:20:00', 1),
    ('3333 Green Valley, City L', 'Convenience Store, City L', 25.60, '2023-09-04 18:50:00', 0),
    ('3434 Sunset Blvd, City I', 'Luxury Store, City I', 1600.00, '2023-09-05 14:10:00', 1),
    ('3535 Country Lane, City J', 'Hardware Store, City J', 68.40, '2023-09-06 09:30:00', 0),
    ('3636 Lakeside View, City K', 'Store Z, City K', 135.75, '2023-09-07 17:20:00', 0),
    ('3737 Forest Glade, City L', 'Cafe E, City L', 38.50, '2023-09-08 12:40:00', 1),
    ('3838 Meadow Lane, City I', 'Online Shop', 280.50, '2023-09-09 08:50:00', 0),
    ('3939 Creekside Ave, City J', 'Grocery Store, City J', 48.75, '2023-09-10 15:30:00', 0),
    ('4040 River Rd, City K', 'Electronics Store, City K', 720.25, '2023-09-11 11:10:00', 1),
    ('4141 Skyway Blvd, City L', 'Boutique, City L', 165.90, '2023-09-12 17:40:00', 0),
    ('4242 Hillcrest Rd, City I', 'Convenience Store, City I', 22.50, '2023-09-13 10:20:00', 0),
    ('4343 Riverbank View, City J', 'Luxury Store, City J', 2100.75, '2023-09-14 16:50:00', 1),
    ('4444 Country Club, City K', 'Hardware Store, City K', 92.30, '2023-09-15 12:30:00', 0),
    ('4545 Lakeside Ave, City L', 'Store Alpha, City L', 145.60, '2023-09-16 08:40:00', 0),
    ('4646 Forest Edge, City I', 'Cafe F, City I', 55.80, '2023-09-17 15:20:00', 1),
    ('4747 Creekside View, City J', 'Online Shop', 320.40, '2023-09-18 11:50:00', 0),
    ('4848 Meadow Park, City K', 'Grocery Store, City K', 62.50, '2023-09-19 18:30:00', 0),
    ('4949 River Walk, City L', 'Electronics Store, City L', 550.30, '2023-09-20 14:10:00', 1);
```

Preparing Data for an Unsupervised Anomaly Detection Model for Logs

The anomaly detection model for logs looks for unusual patterns in log data. The model uses unsupervised learning, so the `target` column is excluded for training and predicting anomalies.

To prepare the data for the anomaly detection model for logs:

1. Connect to the MySQL Server.

2. If not already done, create and use the database to store the data.

```
mysql> CREATE DATABASE anomaly_log_data;
mysql> USE anomaly_log_data;
```

3. Create the table to insert the sample data into. This is the training dataset.

```
mysql> CREATE TABLE training_data (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    log_message TEXT,
    timestamp DATETIME,
    target TINYINT
```

```
);
```

4. Insert the sample data to be trained into the table. Copy and paste the following commands.

```
INSERT INTO training_data (log_message, timestamp, target) VALUES
    ("User login successful: admin", "2023-08-07 09:00:00", 0),
    ("Database connection established", "2023-08-07 09:05:23", 0),
    ("Failed login attempt from IP: 192.168.1.20", "2023-08-07 09:12:15", 1),
    ("Server load is high: 85%", "2023-08-07 09:20:30", 1),
    ("Normal system behavior", "2023-08-07 09:35:00", 0),
    ("Anomalous CPU usage spike", "2023-08-07 10:10:45", 1),
    ("New user registered", "2023-08-07 10:25:00", 0),
    ("Error: File not found", "2023-08-07 11:02:10", 1),
    ("System startup completed", "2023-08-07 11:30:00", 0),
    ("Network packet loss detected", "2023-08-07 12:15:35", 1),
    ("User activity: John accessed dashboard", "2023-08-07 13:00:20", 0),
    ("Security alert: Brute force attack detected", "2023-08-07 13:45:55", 1),
    ("Log rotation completed", "2023-08-07 14:20:00", 0),
    ("Anomalous memory usage pattern", "2023-08-07 15:05:30", 1),
    ("User feedback submitted", "2023-08-07 15:40:10", 0),
    ("System error: Out of memory", "2023-08-07 16:15:25", 1),
    ("Network connectivity restored", "2023-08-07 16:50:00", 0),
    ("Unlabeled log entry", NULL, NULL),
    ("Potential intrusion detected", "2023-08-07 17:35:40", 1),
    ("User logout: Jane", "2023-08-07 18:10:00", 0);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset, but the target column, `target`, must be excluded when generating predictions.

```
mysql> CREATE TABLE testing_data (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    log_message TEXT,
    timestamp DATETIME,
    target TINYINT
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO testing_data (log_message, timestamp, target) VALUES
    ("User login failed: Invalid credentials", "2023-08-08 10:30:00", 1),
    ("Server response time increased", "2023-08-08 11:15:45", 1),
    ("Normal database query", "2023-08-08 12:00:20", 0),
    ("Unusual network traffic from IP: 10.0.0.5", "2023-08-08 12:45:30", 1),
    ("System update completed successfully", "2023-08-08 13:30:00", 0),
    ("Error log: Stack trace included", "2023-08-08 14:10:50", 1),
    ("User activity: Admin accessed settings", "2023-08-08 15:00:10", 0),
    ("Unlabeled log: Further investigation needed", NULL, NULL),
    ("Security alert: Potential malware detected", "2023-08-08 16:25:35", 1),
    ("System shutdown initiated", "2023-08-08 17:10:00", 0);
```

What's Next

- Learn how to Train an Anomaly Detection Model.

4.6.4.5 Training an Anomaly Detection Model

After preparing the data for an anomaly detection model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Anomaly Detection Training
- Anomaly Detection Options

- Semi-supervised Learning Options
- Log Anomaly Detection Options
- Unsupported Anomaly Detection Options
- Unsupported Routines
- Training a Semi-Supervised Anomaly Detection Model
- Training an Unsupervised Anomaly Detection Model for Logs
- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for an Anomaly Detection Model.

Requirements for Anomaly Detection Training

Consider the following based on the type of anomaly detection you are running:

- Set the `task` parameter to `anomaly_detection` for running anomaly detection on table data, or `log_anomaly_detection` for running anomaly detection on log data.
- If running an unsupervised model, the `target_column_name` parameter must be set to `NULL`.
- If running a semi-supervised model:
  - The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows are used to train the unsupervised component, while the rows with a value different than NULL are used to train the supervised component.
  - The `experimental` option must be set to `semisupervised`.
- If running anomaly detection on log data, the input table can only have the following columns:
  - The column containing the logs.
  - If including logs from different sources, a column containing the source of each log. Identify this column with the `log_source_column` option.
  - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
  - At least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
  - If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option to exclude irrelevant columns.

Anomaly Detection Options

Use the following `JSON` *`options`*:

- `contamination`: Represents an estimate of the percentage of outliers in the training table.
  - The contamination factor is calculated as: estimated number of rows with anomalies/total number of rows in the training table.
  - The contamination value must be greater than 0 and less than 0.5. The default value is 0.01.
- `model_list`: Allows you to select the model for training. If no option is specified, the default model is Generalized kth Nearest Neighbors (GkNN). Selecting more than one model or an unsupported model produces an error. Review supported Anomaly Detection Models.

Semi-supervised Learning Options

You have the following options to train a semi-supervised anomaly detection model:

- `supervised_submodel_options`: Allows you to set optional override parameters for the supervised model component. The only model supported is `DistanceWeightedKNNClassifier`. The following parameters are supported:
  - `n_neighbors`: Sets the desired k value that checks the k closest neighbors for each unclassified point. The default value is 5 and the value must be an integer greater than 0.
  - `min_labels`: Sets the minimum number of labeled data points required to train the supervised component. If fewer labeled data points are provided during training of the model, `ML_TRAIN` fails. The default value is 20 and the value must be an integer greater than 0.
- `ensemble_score`: This option specifies the metric to use to score the ensemble of unsupervised and supervised components. It identifies the optimal weight between the two components based on the metric. The supported metrics are `accuracy`, `precision`, `recall`, and `f1`. The default metric is `f1`.

Log Anomaly Detection Options

You have the following options for anomaly detection on log data. The options are available as a separate `JSON_OBJECT` named `logad_options`:

- `additional_masking_regex`: Allows you to mask log data by using regular expression in a `JSON_ARRAY`. By default, the following parameters are automatically masked during training and when generating anomaly scores.
  - IP
  - DATETIME
  - TIME
  - HEX
  - IPPORT
  - OCID
- `window_size`: Specifies the maximum number of log lines to be grouped for anomaly detection. The default value is 10.
- `window_stride`: Specifies the stride value to use for segmenting log lines. For example, there is log A, B, C, D, and E. The `window_size` is 3, and the `window_stride` is 2. The first row has log A, B, and C. The second row has log C, D, and E. If this value is equal to `window_size`, there is no overlapping of log segments. The default value is 3.

- `log_source_column`: Specifies the column name that contains the source identifier of the respective log lines. Log lines are grouped according to their respective source (for example, logs from multiple MySQL databases that are in the same table). By default, all log lines are assumed to be from the same source.

Unsupported Anomaly Detection Options

The following options are not supported for anomaly detection:

- `exclude_model_list`
- `optimization_metric`

Unsupported Routines

You cannot run the following routines for a trained anomaly detection model:

- `ML_EXPLAIN`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`
- `ML_PREDICT_ROW` (only for anomaly detection for logs)

Training a Semi-Supervised Anomaly Detection Model

Train the model with the `ML_TRAIN` routine and use the `credit_card_train` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @semi_supervised_model='anomaly_detection_semi_supervised_use_case';
   ```

   The model handle is set to `anomaly_detection_semi_supervised_use_case`.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_handle
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

   The following example runs `ML_TRAIN` on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('anomaly_data.credit_card_train', "target", CAST('{"task": "anomaly_detection", "e
   ```

   Where:

   - `anomaly_data.credit_card_train` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
   - `target` is the name of the target column, which contains ground truth values to use for semi-supervised learning.

- `CAST('{"task": "anomaly_detection", "experimental": {"semisupervised": {}}}' as JSON)` specifies the machine learning task type. The `experimental` parameter is required to use a semi-supervised learning model. All default values are used for semi-supervised learning.
- `@semi_supervised_model` is the session variable previously set that defines the model handle to the name defined by the user: `anomaly_detection_semi_supervised_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@semi_supervised_model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG  WHERE model_
+----------+----------------------------------------------+-----------------------------------+
| model_id | model_handle                                 | train_table_name                  |
+----------+----------------------------------------------+-----------------------------------+
|        3 | anomaly_detection_semi_supervised_use_case   | anomaly_data.credit_card_train    |
+----------+----------------------------------------------+-----------------------------------+
```

Training an Unsupervised Anomaly Detection Model for Logs

Train the model with the `ML_TRAIN` routine and use the `training_data` table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

```
mysql> SET @variable = 'model_handle';
```

Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

```
mysql> SET @unsupervised_log_model='anomaly_detection_log_use_case';
```

The model handle is set to `anomaly_detection_log_use_case`.

2. Run the `ML_TRAIN` routine.

```
mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
```

Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`model_handle`* with your own values.

The following example runs `ML_TRAIN` on the training dataset previously created.

```
mysql> CALL sys.ML_TRAIN('anomaly_log_data.training_data', NULL, JSON_OBJECT('task', 'log_anomaly_detec
      JSON_ARRAY('log_id', 'timestamp', 'target')), @unsupervised_log_model);
```

Where:

- `anomaly_log_data.training_data` is the fully qualified name of the table that contains the training dataset (*`database_name.table_name`*).
- `NULL` is set for the target column because it is an unsupervised learning model, so no labeled data is used to train the model.

- `JSON_OBJECT('task', 'log_anomaly_detection'` specifies the machine learning task type.
- `'exclude_column_list', JSON_ARRAY('log_id', 'timestamp', 'target')` sets the required options to run the model for anomaly detection on logs. The columns `log_id` and `timestamp` are excluded because they are not any of the required columns for training. See Requirements for Anomaly Detection Training to learn more. The `target` column is excluded because it is an unsupervised learning model.
- `@unsupervised_log_model` is the session variable previously set that defines the model handle to the name defined by the user: `anomaly_detection_log_use_case`. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the `@unsupervised_log_model` session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *`user1`* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG  WHERE model_hand
+----------+-----------------------------------------------+-------------------------------------+
| model_id | model_handle                                  | train_table_name                    |
+----------+-----------------------------------------------+-------------------------------------+
|        4 | anomaly_detection_log_use_case                | anomaly_log_data.training_data      |
+----------+-----------------------------------------------+-------------------------------------+
```

What's Next

- Learn how to Generate Predictions for an Anomaly Detection Model
- Review additional Syntax Examples for Anomaly Detection Training

4.6.4.6 Generating Predictions for an Anomaly Detection Model

After training the model, you can generate predictions.

To generate predictions, use the sample data from the two anomaly detection datasets: `credit_card_train` and `training_data`. Both datasets have labeled and unlabeled rows, but only the dataset for semi-supervised learning uses this for training. The other dataset for log data is trained using unsupervised learning. Having labels for both datasets allows you to compare the predictions to the actual values and determine if the predictions are reliable. Once you determine the trained model is reliable for generating predictions, you can start using it on unseen data.

Anomaly detection models produce anomaly scores, which represent the degree to which a data point deviates from the expected normal behavior. Higher scores indicate a greater degree of abnormality, potentially signaling an anomaly that warrants further investigation. In the results, `is_anomaly` generates a value of 1 for an anomaly, or 0 for normal. The `normal` value represents the degree to which the row of data or log segment exhibits normal behavior. The `anomaly` value represents the degree to which the row of data or log segment exhibits anomalous behavior.

To detect anomalies, run the `ML_PREDICT` routines on data with the same columns as the training model.

- To detect anomalies in row data, you can run the `ML_PREDICT_ROW` or `ML_PREDICT_TABLE` routines.
- To detect anomalies in log data, you can only run the `ML_PREDICT_TABLE` routine.

This topic has the following sections.

Before You Begin

Complete the following tasks:

- Prepare Data for an Anomaly Detection Model
- Train an Anomaly Detection Model.

Requirements for Generating Predictions

If you run `ML_PREDICT_TABLE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Anomaly Detection Model Options

The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly.

There are two ways to set threshold values for anomaly detection models.

Set the Contamination Value

You can set the `contamination` option for the `ML_TRAIN` routine. This option uses the following calculation to set the threshold: (1 - `contamination`)-th percentile of all the anomaly scores.

The default `contamination` value is 0.01. The default `threshold` value based on the default `contamination` value is the 0.99-th percentile of all the anomaly scores.

Set the Threshold Value

You can set the `threshold` option for the `ML_PREDICT_TABLE`, `ML_PREDICT_ROW`, and `ML_SCORE` routines. The value must be greater than 0 and less than 1.

If no value is set for the `threshold` option, the value set for the `contamination` option in the `ML_TRAIN` routine determines the threshold.

The following additional options are available:

- An alternative to `threshold` is `topk`. The results include the top K rows with the highest anomaly scores. The `ML_PREDICT_TABLE` and `ML_SCORE` routines include the `topk` option, which is an integer between 1 and the table length.
- `ML_SCORE` includes an options parameter in `JSON` format. The options are `threshold` and `topk`.
- When running a semi-supervised model, the `ML_PREDICT_ROW`, `ML_PREDICT_TABLE`, and `ML_SCORE` routines have the `supervised_submodel_weight` option. It allows you to override the

`ensemble_score` weighting estimated during `ML_TRAIN` with a new value. The value must be greater than 0 and less than 1.

Options for Anomaly Detection on Log Data

When you run anomaly detection on log data, you have the option to leverage the GenAI feature of MySQL AI for textual summaries of the results. To create summaries, use the following options:

- `summarize_logs`: Enable summaries by setting this to `TRUE`. If enabled, summaries are generated for log segments that are labeled as an anomaly or exceed the value set for the `summary_threshold`.
- `summary_threshold`: Determines the rows in the output table that are summarized. This does not affect how the `contamination` and `threshold` options determine anomalies. You can set a value greater than 0 and less than 1. The default value is `NULL`.

  Summaries are generated for the following:

  - All rows labeled as anomalies.
  - If a value is set for `summary_threshold`, any non-anomaly rows that exceed the value of the `summary_threshold`.

  If the default `NULL` value is used for `summary_threshold`, then only rows labeled as anomalies are summarized.

> **Note**
>
> Enabling the `summary_threshold` option and setting a very low threshold value can potentially lead to a high number of summaries being generated, which may substantially increase the time required to generate output tables.

Generating Predictions for a Semi-Supervised Anomaly Detection Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_semi_supervised_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('anomaly_data.credit_card_train', 'anomaly_detection_semi_supervised_use_c
   ```

   Where:

   - `anomaly_data.credit_card_train` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.

- `anomaly_data.credit_card_predictions_semi` is the fully qualified name of the output table with predictions (*`database_name.table_name`*).
- `JSON_OBJECT('threshold', 0.55)` sets a threshold value of 55%, which means any row that generates an anomaly score of over 55% is labeled as an anomaly.

3. Query the `target` and `ml_results` columns from the output table. This allows you to compare the real value with the generated anomaly prediction. Review `is_anomaly` to see if the row is labeled as an anomaly (1) or normal (0). Review the anomaly score for each prediction next to `normal` and `anomaly`. If needed, you can also query all the columns from the table (`SELECT * FROM credit_card_predictions_semi`) to review all the data at once.

```
mysql> SELECT target, ml_results FROM credit_card_predictions_semi;
+--------+------------------------------------------------------------------------------------------+
| target | ml_results                                                                               |
+--------+------------------------------------------------------------------------------------------+
|      0 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.43, "anomaly": 0.57}}     |
|      1 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4377, "anomaly": 0.5623}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8677, "anomaly": 0.1323}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8652, "anomaly": 0.1348}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4921, "anomaly": 0.5079}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8487, "anomaly": 0.1513}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.7622, "anomaly": 0.2378}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.57, "anomaly": 0.43}}     |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8317, "anomaly": 0.1683}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8539, "anomaly": 0.1461}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9264, "anomaly": 0.0736}} |
|      1 | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4079, "anomaly": 0.5921}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8379, "anomaly": 0.1621}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.7971, "anomaly": 0.2029}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4623, "anomaly": 0.5377}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8816, "anomaly": 0.1184}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8267, "anomaly": 0.1733}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8816, "anomaly": 0.1184}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4661, "anomaly": 0.5339}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8202, "anomaly": 0.1798}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9113, "anomaly": 0.0887}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5078, "anomaly": 0.4922}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9378, "anomaly": 0.0622}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8963, "anomaly": 0.1037}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5262, "anomaly": 0.4738}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.5002, "anomaly": 0.4998}} |
|   NULL | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8767, "anomaly": 0.1233}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.8878, "anomaly": 0.1122}} |
|      1 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.4661, "anomaly": 0.5339}} |
|      0 | {"predictions": {"is_anomaly": 0}, "probabilities": {"normal": 0.9037, "anomaly": 0.0963}} |
|   NULL | {"predictions": {"is_anomaly": 1}, "probabilities": {"normal": 0.4171, "anomaly": 0.5829}} |
+--------+------------------------------------------------------------------------------------------+
```

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

Generating Predictions for an Unsupervised Anomaly Detection Model on Log Data

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

```
mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_log_use_case', NULL);
```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *table_name*, *model_handle*, and *output_table_name* with your own values. Add *options* as needed.

The following example runs ML_PREDICT_TABLE on the testing dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('anomaly_log_data.testing_data', 'anomaly_detection_log_use_case', 'anomal
       JSON_OBJECT('logad_options', JSON_OBJECT('summarize_logs', TRUE)));
```

Where:

- anomaly_log_data.testing_data is the fully qualified name of the input table that contains the data to generate predictions for (*database_name.table_name*).
- @model is the session variable for the model handle.
- anomaly_log_data.log_predictions_unsupervised is the fully qualified name of the output table with predictions (*database_name.table_name*).
- JSON_OBJECT('logad_options', JSON_OBJECT('summarize_logs', TRUE)) enables the textual summaries generated by the GenAI feature of MySQL AI. No threshold is set for the summaries, so the default value of any labeled anomaly generates a summary.

3. Query the output table and compare the real value with the generated anomaly prediction. Use \G to view the output in an easily readable format.

```
mysql> SELECT * FROM log_predictions_unsupervised\G
*************************** 1. row ***************************
                id: 1
parsed_log_segment: User login failed: Invalid credentials Server response time increased Normal database c
        ml_results: {"summary": "\nHere is a concise summary of the text:\n\nThe system encountered several
*************************** 2. row ***************************
                id: 2
parsed_log_segment: Unusual network traffic from IP: 10.0.0.5 System update completed successfully Error lo
        ml_results: {"summary": "\nHere is a concise summary:\n\nA system update was completed successfully
*************************** 3. row ***************************
                id: 3
parsed_log_segment: User activity: Admin accessed settings Unlabeled log: Further investigation needed Secu
        ml_results: {"summary": "\nAn administrator has accessed the system settings, triggered a security
*************************** 4. row ***************************
                id: 4
parsed_log_segment: System shutdown initiated
        ml_results: {"summary": "\nThe system is shutting down.", "index_map": [10], "predictions": {"is_an
```

The size of the output table is based on the window_size and window_stride parameters when the model is trained. Since this use case does not set these parameters, the default values of 10 for window_size and 3 for window_stride is used. See Log Anomaly Detection Options to learn more.

Review the following in the output table:

- is_anomaly to see if the row is labeled as an anomaly (1) or normal (0).
- normal and anomaly to see the anomaly score for each.
- index_map to see which rows in the input table are included in the prediction based on the window_size and window_stride.
- summary to see the generated text summary describing the anomaly.

What's Next

- Learn how to Score an Anomaly Detection Model.

4.6.4.7 Scoring an Anomaly Detection Model

After generating predictions, you can score the model to assess its reliability. For a list of scoring metrics you can use with anomaly detection models, see Anomaly Detection Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

To generate a score, the `target_column_name` column must only contain the anomaly scores as an integer: `1` for an anomaly, or `0` for normal.

Before You Begin

Complete the following tasks:

- Prepare Data for an Anomaly Detection Model
- Train an Anomaly Detection Model
- Generate Predictions for an Anomaly Detection Model

Requirements for Scoring Models

If you run `ML_SCORE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Scoring a Semi-Supervised Anomaly Detection Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_semi_supervised_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `accuracy` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('anomaly_data.credit_card_test', 'target', 'anomaly_detection_semi_supervised_
   ```

   Where:

   - `anomaly_data.credit_card_test` is the fully qualified name of the validation dataset.
   - `target` is the target column name with ground truth values.

- 'anomaly_detection_semi_supervised_use_case' is the model handle for the trained model.
- accuracy is the selected scoring metric.
- @anomaly_score is the session variable name for the score value.
- NULL means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @anomaly_score;
+--------------------+
| @anomaly_score     |
+--------------------+
| 0.6499999761581421 |
+--------------------+
1 row in set (0.0481 sec)
```

4. If done working with the model, unload it with the ML_MODEL_UNLOAD routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('anomaly_detection_semi_supervised_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

Scoring an Unsupervised Anomaly Detection Model for Log Data

Even though you score an unsupervised model, you must provide a labeled dataset for generating a score.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to NULL.

The following example uses the session variable.

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

The following example uses the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD('anomaly_detection_log_use_case', NULL);
```

2. Score the model with the ML_SCORE routine and use the accuracy metric.

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

Replace *table_name*, *target_column_name*, *model_handle*, *metric*, *score* with your own values.

The following example runs ML_SCORE on the testing dataset previously created.

```
mysql> CALL sys.ML_SCORE('anomaly_log_data.testing_data', 'target', 'anomaly_detection_log_use_case', 'f1',
```

Where:

- anomaly_log_data.testing_data is the fully qualified name of the validation dataset.
- target is the target column name with ground truth values.
- 'anomaly_detection_log_use_case' is the model handle for the trained model.

- `f1` is the selected scoring metric.
- `@anomaly_log_score` is the session variable name for the score value.
- `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @anomaly_log_score;
+--------------------+
| @anomaly_log_score |
+--------------------+
| 0.8571428656578064 |
+--------------------+
1 row in set (0.0452 sec)
```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('anomaly_detection_log_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

4.6.5 Generating Recommendations

Recommendation models find patterns in user behavior to recommend products and users based on prior behavior and preferences. Common examples include a streaming service recommending movies and shows based on past viewing history, or an online shopping site recommending products based on prior purchases.

The main goal of recommendation models is to recommend either items that a user will like, or recommend users who may like a specific item. AutoML includes recommendation models that can recommend the following:

- The rating that a user will give to an item.
- Users who will like an item.
- Items that a user will like.
- Identify similar items.
- Identify similar users.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The recommendation use-case is to create a machine learning model based on users giving a rating of 1 to 10 for different items.

To generate your own datasets to create machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.5.1 Recommendation Task Types

You can create recommendation models based on either explicit or implicit feedback. See Recommendation Models to review models that support either implicit or explicit feedback.

Recommendation Models with Explicit Feedback

Recommendation models that use explicit feedback collect data on users that directly provide ratings on items. The user ratings can be positive or negative. The recommendation models then use the feedback to generate predicted ratings for users and items. The ratings are specific values, and the higher the value, the better the rating.

Recommendation Models with Implicit Feedback

Recommendation models that use implicit feedback collect data on users' behavior, such as past purchases, clicks, and view times. Users do not have to explicitly express their taste about an item. When a user interacts with an item, the implication is that they prefer it to an item that they do not interact with. Therefore, only positive observations are available. The non-observed user-item interactions are a blend of negative feedback (the user doesn't like the item) or missing values (the user might be interested in the item). The recommendation model generates rankings for users and items. Rankings are a comparative measure, and the lower the value, the better the ranking. Because A is better than B, the ranking for A has a lower value than the ranking for B. AutoML derives rankings based on ratings from implicit feedback for all ratings that are at or above the feedback threshold.

Implicit feedback data can be in the following formats:

- Unary data: Only records if an interaction occurred or not. This type of data often uses a value of 1 to represent an interaction, such as a click or view. Non-interactions can be represented by a value of 0 or missing values.
- Binary data: Explicitly categorizes interactions as positive or negative, such as users expressing likes or dislikes.
- Numerical data: Provides more granular information about the interaction, such as how long a user watched a video or how many times a user listened to a song. If numerical data is used for implicit feedback, it is important to set the `feedback_threshold` option during training to distinguish what constitutes positive feedback. This threshold determines what value is equivalent to a positive interaction. For example, if users are tracked by how many times they have interacted with an item, you might set the `feedback_threshold` with a value of 3, which means that positive feedback is represented by users that interact with the item more than three times.

Content-Based Recommendation Models

Content-based recommendation models allow you to include item descriptions in the input of the recommendation model. This helps the model provide more accurate representations of items. Currently, content-based recommendation models can only be used with implicit feedback. When training a content-based recommendation model, you can use the Collaborative Topic Regression (CTR) model, which combines the ideas of matrix factorization models and topic modeling using Latent Dirichlet Allocation (LDA).

What's Next

- Learn how to Prepare Data for a Recommendation Model.

4.6.5.2 Preparing Data for a Recommendation Model

This topic describes how to prepare the data to use for a recommendation machine learning model using explicit feedback. It uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a training dataset and a testing dataset. The training dataset has 86 records, and the testing dataset has 40 records. In a real-life use case, you should prepare a larger amount of records for training and testing, and ensure the predictions are valid and reliable before testing on unlabeled data. To ensure reliable predictions, you should create an additional validation dataset. You can reserve 20% of the records in the training dataset to create the validation dataset.

Before You Begin

- Learn how to Prepare Data.

Preparing Data

To prepare the data for the recommendation model:

1. Connect to the MySQL Server.

2. Create and use the database to store the data.

```
mysql> CREATE DATABASE recommendation_data;
mysql> USE recommendation_data;
```

3. Create the table to insert the sample data into. This is the training dataset. The columns for users and items (`user_id` and `item_id`), must be in string data type.

```
mysql> CREATE TABLE training_dataset (
    user_id VARCHAR(3),
    item_id VARCHAR(3),
    rating DECIMAL(3, 1),
    PRIMARY KEY (user_id, item_id)
);
```

4. Insert the sample data to train into the table. Copy and paste the following commands.

```
INSERT INTO training_dataset (user_id, item_id, rating) VALUES
    (1, 1, 5.0),
    (1, 3, 8.0),
    (1, 5, 2.5),
    (1, 7, 6.5),
    (1, 9, 4.0),
    (1, 11, 7.5),
    (1, 13, 3.0),
    (1, 15, 9.0),
    (1, 17, 1.5),
    (1, 19, 5.5),
    (2, 2, 4.5),
    (2, 4, 7.5),
    (2, 6, 2.0),
    (2, 8, 5.5),
    (2, 10, 9.0),
    (2, 12, 3.5),
    (2, 14, 6.0),
    (2, 16, 1.0),
    (2, 18, 4.5),
    (2, 20, 8.5),
    (3, 1, 3.5),
    (3, 4, 6.5),
    (3, 7, 2.5),
    (3, 9, 5.0),
    (3, 11, 8.5),
    (3, 13, 1.0),
    (3, 15, 4.0),
    (3, 17, 7.0),
    (3, 19, 2.5),
```

```
    (4, 2, 5.5),
    (4, 5, 8.5),
    (4, 8, 3.0),
    (4, 10, 6.5),
    (4, 12, 9.5),
    (4, 14, 2.0),
    (4, 16, 4.5),
    (4, 18, 7.5),
    (5, 3, 7.0),
    (5, 6, 1.5),
    (5, 8, 4.0),
    (5, 11, 6.0),
    (5, 13, 8.0),
    (5, 15, 2.5),
    (5, 17, 5.5),
    (5, 19, 9.0),
    (6, 1, 4.5),
    (6, 4, 7.5),
    (6, 6, 3.0),
    (6, 9, 5.5),
    (6, 12, 8.0),
    (6, 14, 1.5),
    (6, 16, 4.0),
    (6, 18, 6.5),
    (7, 2, 6.0),
    (7, 5, 3.5),
    (7, 7, 5.0),
    (7, 10, 7.5),
    (7, 12, 2.0),
    (7, 14, 4.5),
    (7, 16, 7.0),
    (7, 18, 9.5),
    (8, 3, 8.5),
    (8, 6, 2.5),
    (8, 8, 5.0),
    (8, 11, 3.5),
    (8, 13, 6.5),
    (8, 15, 1.0),
    (8, 17, 4.5),
    (8, 19, 7.0),
    (9, 2, 5.0),
    (9, 5, 8.0),
    (9, 7, 1.5),
    (9, 10, 4.0),
    (9, 12, 6.5),
    (9, 14, 9.0),
    (9, 16, 2.5),
    (9, 18, 5.5),
    (10, 1, 6.5),
    (10, 4, 3.0),
    (10, 6, 5.5),
    (10, 8, 8.0),
    (10, 11, 2.0),
    (10, 13, 4.5),
    (10, 15, 7.0),
    (10, 17, 9.5),
    (10, 19, 1.5);
```

5. Create the table to use for generating predictions. This is the test dataset. It has the same columns as the training dataset.

```
mysql> CREATE TABLE testing_dataset (
    user_id VARCHAR(3),
    item_id VARCHAR(3),
    rating DECIMAL(3, 1),
    PRIMARY KEY (user_id, item_id)
);
```

6. Insert the sample data to test into the table. Copy and paste the following commands.

```
INSERT INTO testing_dataset (user_id, item_id, rating) VALUES
    (1, 2, 4.0),
    (1, 4, 7.0),
    (1, 6, 1.5),
    (1, 8, 3.5),
    (2, 1, 5.0),
    (2, 3, 8.0),
    (2, 5, 2.5),
    (2, 7, 6.5),
    (3, 2, 3.5),
    (3, 5, 6.5),
    (3, 8, 2.5),
    (3, 18, 7.0),
    (4, 1, 5.5),
    (4, 3, 8.5),
    (4, 6, 2.0),
    (4, 7, 5.5),
    (5, 2, 7.0),
    (5, 4, 1.5),
    (5, 6, 4.0),
    (5, 12, 5.0),
    (6, 3, 6.0),
    (6, 5, 1.5),
    (6, 7, 4.5),
    (6, 8, 7.0),
    (7, 1, 6.5),
    (7, 4, 3.0),
    (7, 5, 5.5),
    (7, 9, 8.0),
    (8, 2, 8.5),
    (8, 4, 2.5),
    (8, 6, 5.0),
    (8, 9, 3.5),
    (9, 1, 5.0),
    (9, 3, 8.0),
    (9, 7, 2.5),
    (9, 8, 5.5),
    (10, 2, 6.5),
    (10, 5, 3.0),
    (10, 6, 5.5),
    (10, 18, 1.5);
```

What's Next

- Learn how to Train a Recommendation Model.

4.6.5.3 Training a Recommendation Model

After preparing the data for a recommendation model, you can train the model.

This topic has the following sections.

- Before You Begin
- Requirements for Recommendation Training
- Options for Recommendation Models with Explicit Feedback
- Options for Recommendation Models with Implicit Feedback
- Options for Content-Based Recommendation Models
- Unsupported Routines

- Training the Model
- What's Next

Before You Begin

- Review and complete all the tasks to Prepare Data for a Recommendation Model.

Requirements for Recommendation Training

Define the following as required to train a recommendation model.

- Set the `task` parameter to `recommendation` to train a recommendation model.
- `users`: Specifies the column name corresponding to the user IDs. Values in this column must be in a `STRING` data type, otherwise an error is returned during training.
- `items`: Specifies the column name corresponding to the item IDs. Values in this column must be in a `STRING` data type, otherwise an error is returned during training.

If the `users` or `items` column contains `NULL` values, the corresponding rows are dropped and are not be considered during training.

Options for Recommendation Models with Explicit Feedback

Define the following `JSON` options to train a recommendation model with explicit feedback. To learn more about recommendation models, see Recommendation Model Types.

- `feedback`: Set to `explicit`. If not set, the default value is `explicit`.

Options for Recommendation Models with Implicit Feedback

Define the following `JSON` options to train a recommendation model with implicit feedback. To learn more about recommendation models, see Recommendation Model Types.

- `feedback`: Set to `implicit`.
- `feedback_threshold`: The feedback threshold for a recommendation model that uses implicit feedback. It represents the threshold required to be considered positive feedback. For example, if numerical data records the number of times users interact with an item, you might set a threshold with a value of 3. This means users would need to interact with an item more than three times to be considered positive feedback.

Options for Content-Based Recommendation Models

Define the following `JSON` options to train a content-based recommendation model. To learn more about recommendation models, see Recommendation Model Types.

- `item_metadata`: Defines the table that has item description. It is a JSON object that can have the `table_name` option as a key, which specifies the table that has item descriptions. This table must only have two columns: one corresponding to the `item_id`, and the other with a `TEXT` data type (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT) that has the description of the item.
  - `table_name`: To be used with the `item_metadata` option. It specifies the table name that has item descriptions. It must be a string in a fully qualified format (schema_name.table_name) that specifies the table name.

Unsupported Routines

You cannot run the following routines for a trained recommendation model:

- ML_EXPLAIN
- ML_EXPLAIN_ROW
- ML_EXPLAIN_TABLE

Training the Model

Train the model with the ML_TRAIN routine and use the training_data table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *@variable* and *model_handle* with your own definitions. For example:

   ```
   mysql> SET @model='recommendation_use_case';
   ```

   The model handle is set to recommendation_use_case.

2. Run the ML_TRAIN routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
   ```

   Replace *table_name*, *target_column_name*, *task_name*, and *model_handle* with your own values.

   The following example runs ML_TRAIN on the training dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('recommendation_data.training_dataset', 'rating', JSON_OBJECT('task', 'recomme
   ```

   Where:

   - recommendation_data.training_dataset is the fully qualified name of the table that contains the training dataset (*database_name.table_name*).
   - rating is the name of the target column, which contains ground truth values (item ratings).
   - JSON_OBJECT('task', 'recommendation', 'users', 'user_id', 'items', 'item_id') specifies the machine learning task type and defines the users and items columns. Since no model type is defined, the default value of a recommendation model using explicit feedback is trained.
   - @model is the session variable previously set that defines the model handle to the name defined by the user: recommendation_use_case. If you do not define the model handle before training the model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the @model session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG  WHERE model_hand
+----------+-----------------------+------------------------------------+
| model_id | model_handle          | train_table_name                   |
+----------+-----------------------+------------------------------------+
|        5 | recommendation_use_case | recommendation_data.training_dataset |
+----------+-----------------------+------------------------------------+
```

What's Next

- Learn how to Generate Predictions for a Recommendation Model.
- Review additional Syntax Examples for Recommendation Training.

4.6.5.4 Generating Predictions for a Recommendation Model

After training the model, you can generate predictions. To generate predictions, use the sample data from the `testing_dataset` dataset. `NULL` values for any row in the `users` or `items` columns generates an error.

Before You Begin

Complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model

Options for Generating Predictions

The *`options`* for `ML_PREDICT_ROW` and `ML_PREDICT_TABLE` include the following:

- `topk`: The number of recommendations to provide. The default is `3`.
- `recommend`: Specifies what to recommend. Permitted values are:
  - `ratings`: Predicts ratings that users will give. This is the default value.
  - `items`: Recommends items for users.
  - `users`: Recommends users for items.
  - `users_to_items`: This is the same as `items`.
  - `items_to_users`: This is the same as `users`.
  - `items_to_items`: Recommends similar items for items.
  - `users_to_users`: Recommends similar users for users.
- `remove_seen`: If `true`, the model does not repeat existing interactions from the training table. It only applies to the recommendations `items`, `users`, `users_to_items`, and `items_to_users`.

What's Next

- Learn about the different ways to generate specific recommendations with a recommendation model:
  - Generate Predictions for Ratings and Rankings.

- Generate Item Recommendations for Users
- Generate User Recommendations for Items
- Generate Recommendations for Similar Items
- Generate Recommendations for Similar Users

4.6.5.5 Generating Predictions for Ratings and Rankings

This topic describes how to generate recommendations for either ratings (recommendation model with explicit feedback) or rankings (recommendation model with implicit feedback). If generating a rating, the output predicts the rating the user will give to an item. If generating a ranking, the output is a ranking of the user compared to other users.

- For known users and known items, the output includes the predicted rating or ranking for an item for a given pair of `user_id` and `item_id`.
- For a known user with a new item, the prediction is the global average rating or ranking. The routines can add a user bias if the model includes it.
- For a new user with a known item, the prediction is the global average rating or ranking. The routines can add an item bias if the model includes it.
- For a new user with a new item, the prediction is the global average rating or ranking.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generating Rating Recommendations

Since the model you previously trained used explicit feedback, you generate ratings that the user is predicted to give an item. A higher rating means a better rating. If you train a recommendation model using implicit feedback, you generate rankings. A lower ranking means a better ranking. The steps below are the same for both types of recommendation models. See Recommendation Task Types to learn more.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

Replace *table_name*, *model_handle*, and *output_table_name* with your own values. Add *options* as needed.

You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.recomm
```

Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*database_name.table_name*).
- `@model` is the session variable for the model handle.
- `recommendation_data.recommendations` is the fully qualified name of the output table with predictions (*database_name.table_name*).
- `NULL` sets no options for the routine.

3. Query the output table to review the predicted ratings that users give for each user-item pair.

```
mysql> SELECT * from recommendations;
+---------+---------+--------+-----------------------------------+
| user_id | item_id | rating | ml_results                        |
+---------+---------+--------+-----------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"rating": 2.71}} |
| 1       | 4       |    7.0 | {"predictions": {"rating": 3.43}} |
| 1       | 6       |    1.5 | {"predictions": {"rating": 1.6}}  |
| 1       | 8       |    3.5 | {"predictions": {"rating": 2.71}} |
| 10      | 18      |    1.5 | {"predictions": {"rating": 3.63}} |
| 10      | 2       |    6.5 | {"predictions": {"rating": 2.82}} |
| 10      | 5       |    3.0 | {"predictions": {"rating": 3.09}} |
| 10      | 6       |    5.5 | {"predictions": {"rating": 1.67}} |
| 2       | 1       |    5.0 | {"predictions": {"rating": 2.88}} |
| 2       | 3       |    8.0 | {"predictions": {"rating": 4.65}} |
| 2       | 5       |    2.5 | {"predictions": {"rating": 3.09}} |
| 2       | 7       |    6.5 | {"predictions": {"rating": 2.23}} |
| 3       | 18      |    7.0 | {"predictions": {"rating": 3.25}} |
| 3       | 2       |    3.5 | {"predictions": {"rating": 2.53}} |
| 3       | 5       |    6.5 | {"predictions": {"rating": 2.77}} |
| 3       | 8       |    2.5 | {"predictions": {"rating": 2.53}} |
| 4       | 1       |    5.5 | {"predictions": {"rating": 3.36}} |
| 4       | 3       |    8.5 | {"predictions": {"rating": 5.42}} |
| 4       | 6       |    2.0 | {"predictions": {"rating": 1.94}} |
| 4       | 7       |    5.5 | {"predictions": {"rating": 2.61}} |
| 5       | 12      |    5.0 | {"predictions": {"rating": 3.29}} |
| 5       | 2       |    7.0 | {"predictions": {"rating": 2.9}}  |
| 5       | 4       |    1.5 | {"predictions": {"rating": 3.68}} |
| 5       | 6       |    4.0 | {"predictions": {"rating": 1.72}} |
| 6       | 3       |    6.0 | {"predictions": {"rating": 4.98}} |
| 6       | 5       |    1.5 | {"predictions": {"rating": 3.31}} |
| 6       | 7       |    4.5 | {"predictions": {"rating": 2.4}}  |
| 6       | 8       |    7.0 | {"predictions": {"rating": 3.03}} |
| 7       | 1       |    6.5 | {"predictions": {"rating": 3.18}} |
| 7       | 4       |    3.0 | {"predictions": {"rating": 3.95}} |
| 7       | 5       |    5.5 | {"predictions": {"rating": 3.41}} |
| 7       | 9       |    8.0 | {"predictions": {"rating": 3.17}} |
| 8       | 2       |    8.5 | {"predictions": {"rating": 2.6}}  |
| 8       | 4       |    2.5 | {"predictions": {"rating": 3.3}}  |
| 8       | 6       |    5.0 | {"predictions": {"rating": 1.54}} |
```

```
| 8        | 9        |    3.5 | {"predictions": {"rating": 2.65}} |
| 9        | 1        |    5.0 | {"predictions": {"rating": 2.99}} |
| 9        | 3        |    8.0 | {"predictions": {"rating": 4.83}} |
| 9        | 7        |    2.5 | {"predictions": {"rating": 2.32}} |
| 9        | 8        |    5.5 | {"predictions": {"rating": 2.93}} |
+---------+---------+--------+-----------------------------------+
40 rows in set (0.0459 sec)
```

Review each user_id and item_id pair and the respective rating value in the ml_results column. For example, in the first row, user 1 is expected to give item 2 a rating of 2.71.

The values in the rating column refer to the past rating the user_id gave to the item_id. They are not relevant to the values in ml_results.

4. Alternatively, if you do not want to generate an entire table of predicted ratings or rankings, you can run ML_PREDICT_ROW to specify a user-item pair.

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);
```

Replace *input_data* and *model_handle* with your own values. Add *options* as needed.

The following example runs ML_PREDICT_ROW and specifies user 2 and item 1.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id":"2", "item_id": "1"}', @model, NULL);
+-----------------------------------------------------------------------------+
| sys.ML_PREDICT_ROW('{"user_id":"2", "item_id": "1"}', @model, NULL)         |
+-----------------------------------------------------------------------------+
| {"item_id": "1", "user_id": "2", "ml_results": {"predictions": {"rating": 2.88}}} |
+-----------------------------------------------------------------------------+
1 row in set (0.8726 sec)
```

The predicted rating of 2.88 for the user-item pair is the same as the one in the output table previously created.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

4.6.5.6 Generating Item Recommendations for Users

This topic describes how to generate recommended items for users.

- For known users and known items, the output includes a list of items that the user will most likely give a high rating and the predicted rating or ranking.
- For a new user, and an explicit feedback model, the prediction is the global top K items that received the average highest ratings.
- For a new user, and an implicit feedback model, the prediction is the global top K items with the highest number of interactions.

- For a user who has tried all known items, the prediction is an empty list because it is not possible to recommend any other items. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Recommend Items to Users

When you run `ML_PREDICT_TABLE` or `ML_PREDICT_ROW` to generate item recommendations, a default value of three items are recommended. To change this value, set the `topk` parameter.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two items are recommended.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.item_r
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `recommendation_data.item_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
   - `JSON_OBJECT('recommend', 'items', 'topk', 2)` sets the recommendation task to recommend items to users. A maximum of two items to recommend is set.

3. Query the output table to review the recommended top two items for each user in the output table.

```
mysql> SELECT * from item_recommendations;
+---------+---------+--------+--------------------------------------------------------------------+
| user_id | item_id | rating | ml_results                                                         |
+---------+---------+--------+--------------------------------------------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}  |
| 1       | 4       |    7.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}  |
| 1       | 6       |    1.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}  |
| 1       | 8       |    3.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.7, 3.48]}}  |
| 10      | 18      |    1.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}   |
| 10      | 2       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}   |
| 10      | 5       |    3.0 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}   |
| 10      | 6       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.9, 4.65]}}   |
| 2       | 1       |    5.0 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}  |
| 2       | 3       |    8.0 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}  |
| 2       | 5       |    2.5 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}  |
| 2       | 7       |    6.5 | {"predictions": {"item_id": ["3", "17"], "rating": [4.65, 3.38]}}  |
| 3       | 18      |    7.0 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}  |
| 3       | 2       |    3.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}  |
| 3       | 5       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}  |
| 3       | 8       |    2.5 | {"predictions": {"item_id": ["20", "3"], "rating": [4.39, 4.17]}}  |
| 4       | 1       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}  |
| 4       | 3       |    8.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}  |
| 4       | 6       |    2.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}  |
| 4       | 7       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.71, 5.42]}}  |
| 5       | 12      |    5.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}} |
| 5       | 2       |    7.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}} |
| 5       | 4       |    1.5 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}} |
| 5       | 6       |    4.0 | {"predictions": {"item_id": ["20", "18"], "rating": [5.05, 3.74]}} |
| 6       | 3       |    6.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}  |
| 6       | 5       |    1.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}  |
| 6       | 7       |    4.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}  |
| 6       | 8       |    7.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.25, 4.98]}}  |
| 7       | 1       |    6.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}  |
| 7       | 4       |    3.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}  |
| 7       | 5       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}  |
| 7       | 9       |    8.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.41, 5.13]}}  |
| 8       | 2       |    8.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}} |
| 8       | 4       |    2.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}} |
| 8       | 6       |    5.0 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}} |
| 8       | 9       |    3.5 | {"predictions": {"item_id": ["20", "18"], "rating": [4.53, 3.35]}} |
| 9       | 1       |    5.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}  |
| 9       | 3       |    8.0 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}  |
| 9       | 7       |    2.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}  |
| 9       | 8       |    5.5 | {"predictions": {"item_id": ["20", "3"], "rating": [5.09, 4.83]}}  |
+---------+---------+--------+--------------------------------------------------------------------+
40 rows in set (0.0387 sec)
```

Review the recommended items in the `ml_results` column next to `item_id`. For example, user 1 is predicted to like items 20 and 18. Review the ratings in the `ml_results` column to review the expected ratings for each recommended item. For example, user 1 is expected to rate item 20 with a value of 4.7, and item 18 with a value of 3.48.

4. Alternatively, if you do not want to generate an entire table of recommended items, you can run `ML_PREDICT_ROW` to specify a user to recommend items for.

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);
```

Replace *`input_data`* and *`model_handle`* with your own values. Add *`options`* as needed.

The following example runs `ML_PREDICT_ROW` and specifies user 1 with a limit of two recommended items.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "1"}', @model,  JSON_OBJECT('recommend', 'users_to_items'
```

```
+-------------------------------------------------------------------------------------------------------+
| sys.ML_PREDICT_ROW('{"user_id": "1"}', @model,  JSON_OBJECT('recommend', 'users_to_items', 'topk', 2)) |
+-------------------------------------------------------------------------------------------------------+
| {"user_id": "1", "ml_results": {"predictions": {"rating": [4.7, 3.48], "item_id": ["20", "18"]}}}      |
+-------------------------------------------------------------------------------------------------------+
1 row in set (0.7899 sec)
```

The predicted items of 20 and 18 and predicted ratings are the same as the one in the output table previously created.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

4.6.5.7 Generating User Recommendations for Items

This topic describes how to generate recommended users for items.

- For known users and known items, the output includes a list of users that will most likely give a high rating to an item and will also predict the ratings or rankings.
- For a new item, and an explicit feedback model, the prediction is the global top K users who have provided the average highest ratings.
- For a new item, and an implicit feedback model, the prediction is the global top K users with the highest number of interactions.
- For an item that has been tried by all known users, the prediction is an empty list because it is not possible to recommend any other users. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Recommend Users to Items

When you run `ML_PREDICT_TABLE` or `ML_PREDICT_ROW` to generate user recommendations, a default value of three users are recommended. To change this value, set the `topk` parameter.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

The following example uses the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two users are recommended.

```
mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.us
```

Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `recommendation_data.user_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
- `JSON_OBJECT('recommend', 'users', 'topk', 2)` sets the recommendation task to recommend users to items. A maximum of two users to recommend is set.

3. Query the output table to review the recommended top two users for each item in the output table.

```
mysql> SELECT * from user_recommendations;
+---------+---------+--------+--------------------------------------------------------------+
| user_id | item_id | rating | ml_results                                                   |
+---------+---------+--------+--------------------------------------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}   |
| 1       | 4       |    7.0 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}  |
| 1       | 6       |    1.5 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 1       | 8       |    3.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}  |
| 10      | 18      |    1.5 | {"predictions": {"user_id": ["5", "10"], "rating": [3.74, 3.63]}} |
| 10      | 2       |    6.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}   |
| 10      | 5       |    3.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}  |
| 10      | 6       |    5.5 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 2       | 1       |    5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}  |
| 2       | 3       |    8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}  |
| 2       | 5       |    2.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}  |
| 2       | 7       |    6.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}   |
| 3       | 18      |    7.0 | {"predictions": {"user_id": ["5", "10"], "rating": [3.74, 3.63]}} |
| 3       | 2       |    3.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}   |
| 3       | 5       |    6.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}  |
| 3       | 8       |    2.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}  |
| 4       | 1       |    5.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}  |
| 4       | 3       |    8.5 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}  |
| 4       | 6       |    2.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 4       | 7       |    5.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}   |
| 5       | 12      |    5.0 | {"predictions": {"user_id": ["5", "10"], "rating": [3.29, 3.2]}}  |
| 5       | 2       |    7.0 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}   |
```

```
| 5        | 4        |     1.5 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}  |
| 5        | 6        |     4.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 6        | 3        |     6.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}  |
| 6        | 5        |     1.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}  |
| 6        | 7        |     4.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}   |
| 6        | 8        |     7.0 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}  |
| 7        | 1        |     6.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}  |
| 7        | 4        |     3.0 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}  |
| 7        | 5        |     5.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.31, 3.19]}}  |
| 7        | 9        |     8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.34, 3.17]}}  |
| 8        | 2        |     8.5 | {"predictions": {"user_id": ["6", "5"], "rating": [3.02, 2.9]}}   |
| 8        | 4        |     2.5 | {"predictions": {"user_id": ["4", "7"], "rating": [4.16, 3.95]}}  |
| 8        | 6        |     5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [1.94, 1.84]}}  |
| 8        | 9        |     3.5 | {"predictions": {"user_id": ["4", "7"], "rating": [3.34, 3.17]}}  |
| 9        | 1        |     5.0 | {"predictions": {"user_id": ["4", "7"], "rating": [3.36, 3.18]}}  |
| 9        | 3        |     8.0 | {"predictions": {"user_id": ["4", "7"], "rating": [5.42, 5.13]}}  |
| 9        | 7        |     2.5 | {"predictions": {"user_id": ["4", "6"], "rating": [2.61, 2.4]}}   |
| 9        | 8        |     5.5 | {"predictions": {"user_id": ["7", "6"], "rating": [3.12, 3.03]}}  |
+---------+---------+--------+-------------------------------------------------------------------+
40 rows in set (0.0476 sec)
```

Review the recommended users in the `ml_results` column next to `user_id`. For example, for item 2, users 6 and 5 are the top users predicted to like it. Review the ratings in the `ml_results` column to review the expected ratings for each recommended item. For example, user 6 is expected to rate item 2 with a value of 3.02, and user 5 with a value of 2.9.

4. Alternatively, if you do not want to generate an entire table of recommended users, you can run `ML_PREDICT_ROW` to specify an item to recommend items for.

   ```
   mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);
   ```

   Replace *`input_data`* and *`model_handle`* with your own values. Add *`options`* as needed.

   The following example runs `ML_PREDICT_ROW` and specifies item 2 with a limit of two recommended users.

   ```
   mysql> SELECT sys.ML_PREDICT_ROW('{"item_id": "2"}', @model,  JSON_OBJECT('recommend', 'items_to_users', 't
   +----------------------------------------------------------------------------------------------------+
   | sys.ML_PREDICT_ROW('{"item_id": "2"}', @model,  JSON_OBJECT('recommend', 'items_to_users', 'topk', 2)) |
   +----------------------------------------------------------------------------------------------------+
   | {"item_id": "2", "ml_results": {"predictions": {"rating": [3.02, 2.9], "user_id": ["6", "5"]}}}        |
   +----------------------------------------------------------------------------------------------------+
   1 row in set (0.8488 sec)
   ```

   The predicted users of 5 and 6 and predicted ratings are the same as the one in the output table previously created.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate Recommendations for Similar Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

4.6.5.8 Generating Recommendations for Similar Items

This topic describes how to generate recommendations for similar items.

- For known items, the output includes a list of predicted items that have similar ratings and are appreciated by similar users.
- The predictions are expressed in cosine similarity, and range from 0, very dissimilar, to 1, very similar.
- For a new item, there is no information to provide a prediction. This generates an error.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generating Similar Items

When you run `ML_PREDICT_TABLE` or `ML_PREDICT_ROW` to generate similar item recommendations, a default value of three similar items are recommended. To change this value, set the `topk` parameter.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two similar items are generated.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.si
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.

- `recommendation_data.similar_item_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
- `JSON_OBJECT('recommend', 'items_to_items', 'topk', 2)` sets the recommendation task to recommend similar items. A maximum of two similar items is set.

3. Query the output table to review the top two similar items for each item in the output table.

```
mysql> SELECT * from similar_item_recommendations;
+---------+---------+--------+------------------------------------------------------------------------------------+
| user_id | item_id | rating | ml_results                                                                         |
+---------+---------+--------+------------------------------------------------------------------------------------+
| 1       | 2       |    4.0 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}  |
| 1       | 4       |    7.0 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}   |
| 1       | 6       |    1.5 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}  |
| 1       | 8       |    3.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}  |
| 10      | 18      |    1.5 | {"predictions": {"item_id": ["16", "12"], "similarity": [0.9869, 0.9464]}} |
| 10      | 2       |    6.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}  |
| 10      | 5       |    3.0 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}  |
| 10      | 6       |    5.5 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}  |
| 2       | 1       |    5.0 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}} |
| 2       | 3       |    8.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}} |
| 2       | 5       |    2.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}  |
| 2       | 7       |    6.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}} |
| 3       | 18      |    7.0 | {"predictions": {"item_id": ["16", "12"], "similarity": [0.9869, 0.9464]}} |
| 3       | 2       |    3.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}  |
| 3       | 5       |    6.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}  |
| 3       | 8       |    2.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}  |
| 4       | 1       |    5.5 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}} |
| 4       | 3       |    8.5 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}} |
| 4       | 6       |    2.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}  |
| 4       | 7       |    5.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}} |
| 5       | 12      |    5.0 | {"predictions": {"item_id": ["18", "16"], "similarity": [0.9464, 0.9454]}} |
| 5       | 2       |    7.0 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}  |
| 5       | 4       |    1.5 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}   |
| 5       | 6       |    4.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}  |
| 6       | 3       |    6.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}} |
| 6       | 5       |    1.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}  |
| 6       | 7       |    4.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}} |
| 6       | 8       |    7.0 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}  |
| 7       | 1       |    6.5 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}} |
| 7       | 4       |    3.0 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}   |
| 7       | 5       |    5.5 | {"predictions": {"item_id": ["16", "2"], "similarity": [0.9036, 0.8586]}}  |
| 7       | 9       |    8.0 | {"predictions": {"item_id": ["1", "4"], "similarity": [0.7721, 0.6838]}}   |
| 8       | 2       |    8.5 | {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965]}}  |
| 8       | 4       |    2.5 | {"predictions": {"item_id": ["9", "6"], "similarity": [0.6838, 0.6444]}}   |
| 8       | 6       |    5.0 | {"predictions": {"item_id": ["8", "17"], "similarity": [0.8991, 0.8412]}}  |
| 8       | 9       |    3.5 | {"predictions": {"item_id": ["1", "4"], "similarity": [0.7721, 0.6838]}}   |
| 9       | 1       |    5.0 | {"predictions": {"item_id": ["15", "17"], "similarity": [0.8462, 0.7966]}} |
| 9       | 3       |    8.0 | {"predictions": {"item_id": ["19", "13"], "similarity": [0.9826, 0.8851]}} |
| 9       | 7       |    2.5 | {"predictions": {"item_id": ["11", "15"], "similarity": [0.6959, 0.6724]}} |
| 9       | 8       |    5.5 | {"predictions": {"item_id": ["6", "17"], "similarity": [0.8991, 0.7942]}}  |
+---------+---------+--------+------------------------------------------------------------------------------------+
40 rows in set (0.0401 sec)
```

Review the recommended similar items in the `ml_results` column next to `item_id`. For example, for item 2, items 14 and 10 are the top items predicted to be most similar. Review the similarity values in the `ml_results` column next to `similarity` to review the how similar each item is. For example, item 14 has a similarity value of 0.9831 to item 2, and item 10 has a similarity value of 0.965.

4. Alternatively, if you do not want to generate an entire table of similar items, you can run `ML_PREDICT_ROW` to specify an item to recommend similar items for.

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);
```

Replace *`input_data`* and *`model_handle`* with your own values. Add *`options`* as needed.

The following example runs `ML_PREDICT_ROW` and specifies item 2 with a limit of two recommended similar items.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"item_id": "2"}', @model,  JSON_OBJECT('recommend', 'items_to_items'
+------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"item_id": "2"}', @model,  JSON_OBJECT('recommend', 'items_to_items', 'topk', 2)
+------------------------------------------------------------------------------------------------------
| {"item_id": "2", "ml_results": {"predictions": {"item_id": ["14", "10"], "similarity": [0.9831, 0.965
+------------------------------------------------------------------------------------------------------
1 row in set (0.8227 sec)
```

The similar items of 14 and 10 and similarity values are the same as the one in the output table previously created.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items
  - Generate Recommendations for Similar Users
- Learn how to Score a Recommendation Model.

4.6.5.9 Generating Recommendations for Similar Users

This topic describes how to generate recommendations for similar users.

- For known users, the output includes a list of predicted users that have similar behavior and taste.
- The predictions are expressed in cosine similarity, and range from 0, very dissimilar, to 1, very similar.
- For a new user, there is no information to provide a prediction. This generates an error.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model

Generating Similar Users

When you run `ML_PREDICT_TABLE` or `ML_PREDICT_ROW` to generate similar user recommendations, a default value of three similar users are recommended. To change this value, set the `topk` parameter.

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Make predictions for the test dataset by using the `ML_PREDICT_TABLE` routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *`table_name`*, *`model_handle`*, and *`output_table_name`* with your own values. Add *`options`* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs `ML_PREDICT_TABLE` on the testing dataset previously created and sets the `topk` parameter to 2, so only two similar users are generated.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE('recommendation_data.testing_dataset', @model, 'recommendation_data.simila
   ```

   Where:

   - `recommendation_data.testing_dataset` is the fully qualified name of the input table that contains the data to generate predictions for (*`database_name.table_name`*).
   - `@model` is the session variable for the model handle.
   - `recommendation_data.similar_user_recommendations` is the fully qualified name of the output table with recommendations (*`database_name.table_name`*).
   - `JSON_OBJECT('recommend', 'users_to_users', 'topk', 2)` sets the recommendation task to recommend similar users. A maximum of two similar users is set.

3. Query the output table to review the top two similar users generated for each user in the output table.

   ```
   mysql> SELECT * from similar_user_recommendations;
   +---------+---------+--------+-----------------------------------------------------------------------------+
   | user_id | item_id | rating | ml_results                                                                  |
   +---------+---------+--------+-----------------------------------------------------------------------------+
   | 1       | 2       |    4.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}  |
   | 1       | 4       |    7.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}  |
   | 1       | 6       |    1.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}  |
   | 1       | 8       |    3.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]}}  |
   | 10      | 18      |    1.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}  |
   | 10      | 2       |    6.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}  |
   | 10      | 5       |    3.0 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}  |
   | 10      | 6       |    5.5 | {"predictions": {"user_id": ["3", "5"], "similarity": [0.6827, 0.5943]}}  |
   | 2       | 1       |    5.0 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}  |
   | 2       | 3       |    8.0 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}  |
   | 2       | 5       |    2.5 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}  |
   | 2       | 7       |    6.5 | {"predictions": {"user_id": ["7", "9"], "similarity": [0.6473, 0.5746]}}  |
   | 3       | 18      |    7.0 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}} |
   | 3       | 2       |    3.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}} |
   | 3       | 5       |    6.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}} |
   | 3       | 8       |    2.5 | {"predictions": {"user_id": ["1", "10"], "similarity": [0.7922, 0.6827]}} |
   ```

```
| 4        | 1        |      5.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4        | 3        |      8.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4        | 6        |      2.0 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 4        | 7        |      5.5 | {"predictions": {"user_id": ["9", "7"], "similarity": [0.9764, 0.9087]}}
| 5        | 12       |      5.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5        | 2        |      7.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5        | 4        |      1.5 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 5        | 6        |      4.0 | {"predictions": {"user_id": ["8", "1"], "similarity": [0.992, 0.7238]}}
| 6        | 3        |      6.0 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6        | 5        |      1.5 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6        | 7        |      4.5 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 6        | 8        |      7.0 | {"predictions": {"user_id": ["4", "9"], "similarity": [0.5695, 0.4862]}}
| 7        | 1        |      6.5 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7        | 4        |      3.0 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7        | 5        |      5.5 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 7        | 9        |      8.0 | {"predictions": {"user_id": ["9", "4"], "similarity": [0.9738, 0.9087]}}
| 8        | 2        |      8.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8        | 4        |      2.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8        | 6        |      5.0 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 8        | 9        |      3.5 | {"predictions": {"user_id": ["5", "1"], "similarity": [0.992, 0.6356]}}
| 9        | 1        |      5.0 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9        | 3        |      8.0 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9        | 7        |      2.5 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
| 9        | 8        |      5.5 | {"predictions": {"user_id": ["4", "7"], "similarity": [0.9764, 0.9738]}}
+---------+---------+--------+------------------------------------------------------------------------
40 rows in set (0.0414 sec)
```

Review the recommended similar users in the `ml_results` column next to `user_id`. For example, for user 1, users 3 and 5 are the top users predicted to be most similar. Review the similarity values in the `ml_results` column next to `similarity` to review the how similar each user is. For example, user 3 has a similarity value of 0.7922 to user 1, and user 5 has a similarity value of 0.7238.

4. Alternatively, if you do not want to generate an entire table of similar items, you can run `ML_PREDICT_ROW` to specify a user to recommend similar users for.

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);
```

Replace *`input_data`* and *`model_handle`* with your own values. Add *`options`* as needed.

The following example runs `ML_PREDICT_ROW` and specifies user 1 with a limit of two recommended similar users.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "1"}', @model,  JSON_OBJECT('recommend', 'users_to_users'
+-----------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"user_id": "1"}', @model,  JSON_OBJECT('recommend', 'users_to_users', 'topk', 2)
+-----------------------------------------------------------------------------------------------------
| {"user_id": "1", "ml_results": {"predictions": {"user_id": ["3", "5"], "similarity": [0.7922, 0.7238]
+-----------------------------------------------------------------------------------------------------
1 row in set (0.7951 sec)
```

The similar users of 3 and 5 and similarity values are the same as the one in the output table previously created.

What's Next

- Learn how to generate different types of recommendations:
  - Generate Predictions for Ratings and Rankings
  - Generate Item Recommendations for Users
  - Generate User Recommendations for Items

  - Generate Recommendations for Similar Items

- Learn how to Score a Recommendation Model.

4.6.5.10 Scoring a Recommendation Model

After generating predicted ratings/rankings and recommendations, you can score the model to assess its reliability. For a list of scoring metrics you can use with recommendation models, see Recommendation Model Metrics. For this use case, you use the test dataset for validation. In a real-world use case, you should use a separate validation dataset that has the target column and ground truth values for the scoring validation. You should also use a larger number of records for training and validation to get a valid score.

Before You Begin

Review and complete the following tasks:

- Prepare Data for a Recommendation Model
- Train a Recommendation Model
- Generate Predictions for a Recommendation Model
- Generate Predictions for Ratings and Rankings
- Generate Item Recommendations for Users
- Generate User Recommendations for Items
- Generate Recommendations for Similar Items
- Generate Recommendations for Similar Users

Scoring the Model

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to `NULL`.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('recommendation_use_case', NULL);
   ```

2. Score the model with the `ML_SCORE` routine and use the `precision_at_k` metric.

   ```
   mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
   ```

   Replace *`table_name`*, *`target_column_name`*, *`model_handle`*, *`metric`*, *`score`* with your own values.

   The following example runs `ML_SCORE` on the testing dataset previously created.

   ```
   mysql> CALL sys.ML_SCORE('recommendation_data.testing_dataset', 'rating', @model, 'precision_at_k', @recomm
   ```

   Where:

- `recommendation_data.testing_dataset` is the fully qualified name of the validation dataset.
- `rating` is the target column name with ground truth values.
- `@model` is the session variable for the model handle.
- `precision_at_k` is the selected scoring metric.
- `@recommendation_score` is the session variable name for the score value.
- `NULL` means that no other options are defined for the routine.

3. Retrieve the score by querying the @score session variable.

```
mysql> SELECT @recommendation_score;
+-----------------------+
| @recommendation_score |
+-----------------------+
| 0.23333333432674408   |
+-----------------------+
```

4. If done working with the model, unload it with the `ML_MODEL_UNLOAD` routine.

```
mysql> CALL sys.ML_MODEL_UNLOAD('recommendation_use_case');
```

To avoid consuming too much memory, it is good practice to unload a model when you are finished using it.

What's Next

- Review other Machine Learning Use Cases.

4.6.6 Topic Modeling

Topic modeling is an unsupervised machine learning technique that's capable of scanning a set of documents, detecting word and phrase patterns within them, and automatically clustering word groups and similar expressions that best characterize the documents.

The following tasks use a dataset generated by OCI GenAI using Meta Llama Models. The topic modeling use-case is to summarize movie plots.

To generate your own datasets to create machine learning models in MySQL AI, learn how to Generate Text-Based Content.

Note

Datasets were generated using Meta Llama models. Your use of this Llama model is subject to your Oracle agreements and this Llama license agreement: https://downloads.mysql.com/docs/LLAMA_31_8B_INSTRUCT-license.pdf.

4.6.6.1 Preparing Data for Topic Modeling

This topic describes how to prepare the data to use for topic modeling. The model uses a data sample generated by OCI GenAI. To prepare the data for this use case, you set up a dataset to use for both training and testing.

Before You Begin

- Learn how to Prepare Data.

Preparing Data

To prepare the data for topic modeling:

1. Connect to the MySQL Server.
2. Create and use the database to store the data.

```
mysql> CREATE DATABASE topic_modeling_data;
mysql> USE topic_modeling_data;
```

3. Create the table to use for both training and testing.

```
mysql> CREATE TABLE movies ( description TEXT );
```

4. Insert the sample data into the table. Copy and paste the following commands.

```
INSERT INTO movies (description) VALUES ('In a post-apocalyptic wasteland, a lone survivor named Max seeks
INSERT INTO movies (description) VALUES('A young man named Neo discovers that the world as he knows it is a
INSERT INTO movies (description) VALUES('A wealthy family, the Corleones, is drawn into the underworld of N
INSERT INTO movies (description) VALUES('A group of scientists attempt to harness the power of a black hole
INSERT INTO movies (description) VALUES('A young woman, Alice, finds herself in a mysterious and fantastica
INSERT INTO movies (description) VALUES('A young man named Luke Skywalker joins forces with a group of rebe
INSERT INTO movies (description) VALUES('A team of scientists and explorers travel through a wormhole in sp
INSERT INTO movies (description) VALUES('A young Viking named Hiccup aspires to hunt dragons like his tribe
INSERT INTO movies (description) VALUES('A young FBI agent, Clarice Starling, is assigned to help find a se
INSERT INTO movies (description) VALUES('A young man named Harry Potter discovers that he is a wizard and i
INSERT INTO movies (description) VALUES('A group of criminals are given a second chance at redemption by pe
INSERT INTO movies (description) VALUES('A young woman, Elle Woods, is determined to win back her ex-boyfri
INSERT INTO movies (description) VALUES('A group of friends travel to a remote cabin in the woods for a vac
INSERT INTO movies (description) VALUES('A young man named Marty McFly accidentally travels back in time in
INSERT INTO movies (description) VALUES('A young woman, Katniss Everdeen, volunteers to take her younger si
INSERT INTO movies (description) VALUES('A young man named Frodo Baggins inherits a powerful ring, which he
INSERT INTO movies (description) VALUES('A young woman, Jo March, and her sisters come of age in America du
INSERT INTO movies (description) VALUES('A group of astronauts on a mission to Mars face a critical emergen
INSERT INTO movies (description) VALUES('A young man, Scott, discovers a hidden virtual world called the OA
INSERT INTO movies (description) VALUES('A group of high school students from different social cliques are
```

What's Next

- Learn how to Train a Model with Topic Modeling.

4.6.6.2 Training a Model with Topic Modeling

After preparing the data for topic modeling, you can train the model.

Before You Begin

- Review and complete all the tasks to Prepare Data for Topic Modeling.

Requirements for Topic Modeling Training

Define the following required parameters for topic modeling.

- Set the `task` parameter to `topic_modeling`.
- `document_column`: Define the column that contains the text that the model uses to generate topics and tags as output. The output is an array of word groups that best characterize the text.

Unsupported Topic Modeling Options

When the AutoML runs topic modeling, the operation is based on a single algorithm that does not require the tuning of hyperparameters. Moreover, topic modeling is an unsupervised task, which means there are no labels. Therefore, you cannot use the following options for topic modeling:

- model_list
- optimization_metric
- exclude_model_list
- exclude_column_list
- include_column_list

Unsupported Routines

You cannot run the following routines for topic modeling:

- ML_EXPLAIN
- ML_EXPLAIN_ROW
- ML_EXPLAIN_TABLE
- ML_SCORE

Training the Model

Train the model with the ML_TRAIN routine and use the movies table previously created. Before training the model, it is good practice to define the model handle instead of automatically creating one. See Defining Model Handle.

1. Optionally, set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *@variable* and *model_handle* with your own definitions. For example:

   ```
   mysql> SET @model='topic_modeling_use_case';
   ```

   The model handle is set to topic_modeling_use_case.

2. Run the ML_TRAIN routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), model_ha
   ```

   Replace *table_name*, *target_column_name*, *task_name*, and *model_handle* with your own values.

   The following example runs ML_TRAIN on the dataset previously created.

   ```
   mysql> CALL sys.ML_TRAIN('topic_modeling_data.movies', NULL, JSON_OBJECT('task', 'topic_modeling', 'doc
   ```

   Where:

   - topic_modeling_data.movies is the fully qualified name of the table that contains the training dataset (*database_name.table_name*).
   - NULL is set for the target column because topic modeling uses unlabeled data, so you cannot set a target column.
   - JSON_OBJECT('task', 'topic_modeling') specifies the machine learning task type.
   - @model is the session variable previously set that defines the model handle to the name defined by the user: topic_modeling_use_case. If you do not define the model handle before training the

model, the model handle is automatically generated, and the session variable only stores the model handle for the duration of the connection. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted. See Work with Model Handles to learn more.

3. When the training operation finishes, the model handle is assigned to the @model session variable, and the model is stored in the model catalog. View the entry in the model catalog with the following query. Replace *user1* with your MySQL account name.

```
mysql> SELECT model_id, model_handle, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handl
+----------+------------------------+---------------------------+
| model_id | model_handle           | train_table_name          |
+----------+------------------------+---------------------------+
|        6 | topic_modeling_use_case | topic_modeling_data.movies |
+----------+------------------------+---------------------------+
```

What's Next

- Learn how to Generate Predictions for Topic Modeling.

4.6.6.3 Running Topic Modeling on Trained Text

After training the model, you can run topic modeling on the trained text.

To run topic modeling, use the sample data from the movies dataset. The dataset has no target column. When the output table generates, you can review the generated word groups and expressions for the trained text.

Before You Begin

Complete the following tasks:

- Prepare Data for Topic Modeling.
- Train a Model with Topic Modeling

Running Topic Modeling for a Table

1. If not already done, load the model. You can use the session variable for the model that is valid for the duration of the connection. Alternatively, you can use the model handle previously set. For the option to set the user name, you can set it to NULL.

   The following example uses the session variable.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

   The following example uses the model handle.

   ```
   mysql> CALL sys.ML_MODEL_LOAD('topic_modeling_use_case', NULL);
   ```

2. Run topic modeling on the dataset by using the ML_PREDICT_TABLE routine.

   ```
   mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
   ```

   Replace *table_name*, *model_handle*, and *output_table_name* with your own values. Add *options* as needed.

   You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

   The following example runs ML_PREDICT_TABLE on the dataset previously created.

```
mysql> CALL sys.ML_PREDICT_TABLE('topic_modeling_data.movies', @model, 'topic_model
```

Where:

- `topic_modeling_data.movies` is the fully qualified name of the input table that contains the data to run topic modeling for (*`database_name.table_name`*).
- `@model` is the session variable for the model handle.
- `topic_modeling_data.topic_modeling_predictions` is the fully qualified name of the output table with generated word groups and expressions (*`database_name.table_name`*).
- `NULL` sets no options for the routine.

3. Query the `ml_results` column from the output table. Review the generated word groups and expressions for the movie descriptions next to `tags`.

```
mysql> SELECT ml_results FROM topic_modeling_predictions;
+-----------------------------------------------------------------------------------------------------
| ml_results
+-----------------------------------------------------------------------------------------------------
| {"predictions": {"tags": ["dangerous", "future", "join force", "journey", "warrior", "battle", "rebel
| {"predictions": {"tags": ["machine", "world", "agent", "group rebel", "human", "real", "real world",
| {"predictions": {"tags": ["family", "empire", "rival", "criminal", "powerful"]}}
| {"predictions": {"tags": ["scientist", "astronaut", "attempt", "humanity", "massive", "work", "earth"
| {"predictions": {"tags": ["include", "mysterious", "strange", "world", "woman", "young woman", "young
| {"predictions": {"tags": ["empire", "force", "dark force", "evil", "group rebel", "join force", "wise
| {"predictions": {"tags": ["alien", "attempt", "humanity", "planet", "battle", "creature", "ensure", "
| {"predictions": {"tags": ["future", "human", "creature", "form", "learn", "secret", "discover", "youn
| {"predictions": {"tags": ["agent", "deal personal", "murder", "personal", "strange", "victim", "form"
| {"predictions": {"tags": ["wizard", "power", "dark", "learn", "school", "secret", "young", "destroy",
| {"predictions": {"tags": ["alien", "dangerous", "massive", "mission", "planet", "work", "criminal", "
| {"predictions": {"tags": ["murder", "student", "challenge", "help", "school", "face", "woman", "young
| {"predictions": {"tags": ["group", "fall", "friend", "place", "victim", "creature", "survival", "trav
| {"predictions": {"tags": ["fall", "friend", "machine", "secure", "ensure", "scientist", "travel", "de
| {"predictions": {"tags": ["place", "sister", "young", "fight", "woman", "young woman"]}}
| {"predictions": {"tags": ["deal personal", "mysterious", "personal", "secure", "criminal", "powerful"
| {"predictions": {"tags": ["wizard", "dark force", "evil", "include", "journey", "warrior", "wise", "d
| {"predictions": {"tags": ["family", "sister", "challenge", "deal", "woman", "young woman", "young"]}}
| {"predictions": {"tags": ["astronaut", "mission", "earth", "face", "group"]}}
| {"predictions": {"tags": ["world", "real", "real world", "rival", "challenge", "discover", "face", "j
| {"predictions": {"tags": ["student", "form", "learn", "school", "secret", "force", "group"]}}
+-----------------------------------------------------------------------------------------------------
21 rows in set (0.0472 sec)
```

To modify the number of word groups in the ml_results column, you can set the `topk` option. This option must be an integer greater or equal to one. The following example uses the same trained table and adds the option to limit the number of generated word groups to five.

```
mysql> CALL sys.ML_PREDICT_TABLE('topic_modeling_data.movies', @model, 'topic_modeling_data.topic_modeling_
```

Query the `ml_results` column to review the top five generated word groups.

```
mysql> SELECT ml_results FROM topic_modeling_predictions_five;
+-----------------------------------------------------------------------------------------------+
| ml_results                                                                                    |
+-----------------------------------------------------------------------------------------------+
| {"predictions": {"tags": ["dangerous", "future", "join force", "journey", "warrior"]}}       |
| {"predictions": {"tags": ["machine", "world", "agent", "group rebel", "human"]}}             |
| {"predictions": {"tags": ["family", "empire", "rival", "criminal", "powerful"]}}             |
| {"predictions": {"tags": ["scientist", "astronaut", "attempt", "humanity", "massive"]}}      |
| {"predictions": {"tags": ["include", "mysterious", "strange", "world", "woman"]}}            |
| {"predictions": {"tags": ["empire", "force", "dark force", "evil", "group rebel"]}}          |
```

```
| {"predictions": {"tags": ["alien", "attempt", "humanity", "planet", "battle"]}}                  |
| {"predictions": {"tags": ["future", "human", "creature", "form", "learn"]}}                      |
| {"predictions": {"tags": ["agent", "deal personal", "murder", "personal", "strange"]}}           |
| {"predictions": {"tags": ["wizard", "power", "dark", "learn", "school"]}}                        |
| {"predictions": {"tags": ["alien", "dangerous", "massive", "mission", "planet"]}}                |
| {"predictions": {"tags": ["murder", "student", "challenge", "help", "school"]}}                  |
| {"predictions": {"tags": ["group", "fall", "friend", "place", "victim"]}}                        |
| {"predictions": {"tags": ["fall", "friend", "machine", "secure", "ensure"]}}                     |
| {"predictions": {"tags": ["place", "sister", "young", "fight", "woman"]}}                        |
| {"predictions": {"tags": ["deal personal", "mysterious", "personal", "secure", "criminal"]}} |
| {"predictions": {"tags": ["wizard", "dark force", "evil", "include", "journey"]}}                |
| {"predictions": {"tags": ["family", "sister", "challenge", "deal", "woman"]}}                    |
| {"predictions": {"tags": ["astronaut", "mission", "earth", "face", "group"]}}                    |
| {"predictions": {"tags": ["world", "real", "real world", "rival", "challenge"]}}                 |
| {"predictions": {"tags": ["student", "form", "learn", "school", "secret"]}}                      |
+-----------------------------------------------------------------------------------------------+
21 rows in set (0.0475 sec)
```

To learn more about generating predictions for one or more rows of data, see Generate Predictions for a Row of Data.

What's Next

- Review other Machine Learning Use Cases.

4.7 Manage Machine Learning Models

The following sections describe how to manage your machine learning models.

4.7.1 The Model Catalog

AutoML stores machine learning models in a model catalog. A model catalog is a table named `MODEL_CATALOG`. AutoML creates a model catalog for any user that creates a machine learning model.

The `MODEL_CATALOG` table is created in a schema named `ML_SCHEMA_user_name`, where the *`user_name`* is the name of the owning user.

The fully qualified name of the model catalog table is `ML_SCHEMA_user_name.MODEL_CATALOG`.

When a user creates a model, the `ML_TRAIN` routine creates the model catalog schema and table if they do not exist. `ML_TRAIN` inserts the model as a row in the `MODEL_CATALOG` table at the end of training.

A model catalog is accessible only to the owning user unless the user grants privileges on the model catalog to another user. This means that AutoML routines can only use models that are accessible to the user running the routines. For information about giving access to the model catalog and trained models to other users, see Share a Model.

A database administrator can manage a model catalog table the same way as a regular MySQL table.

4.7.1.1 The Model Catalog Table

The `MODEL_CATALOG` table (`ML_SCHEMA_user_name.MODEL_CATALOG`) has the following columns:

- `model_id`

  A primary key, and a unique auto-incrementing numeric identifier for the model.

- `model_handle`

  A name for the model. The model handle must be unique in the model catalog. The model handle is generated or set by the user when the `ML_TRAIN` routine runs on a training dataset. The generated `model_handle` format is *`schemaName_tableName_userName_No`*, as in the following example: `heatwaveml_bench.census_train_`*`user1`*`_1636729526`. See Work with Model Handles to learn more.

Note

The format of the generated model handle is subject to change.

- `model_object`

  Set to null. Models are stored in the `model_object_catalog` table.

- `model_owner`

  The user who initiated the `ML_TRAIN` query to create the model.

- `build_timestamp`

  A timestamp indicating when the model was created (in UNIX epoch time). A model is created when the `ML_TRAIN` routine finishes running.

- `target_column_name`

  The name of the column in the training table that was specified as the target column.

- `train_table_name`

  The name of the input table specified in the `ML_TRAIN` query.

- `model_object_size`

  The model object size, in bytes.

- `model_type`

  The type of model (algorithm) selected by `ML_TRAIN` to build the model.

- `task`

  The task type specified in the `ML_TRAIN` query.

- `column_names`

  The feature columns used to train the model.

- `model_explanation`

  The model explanation generated during training. See Generate Model Explanations.

- `last_accessed`

  The last time the model was accessed. AutoML routines update this value to the current timestamp when accessing the model.

- `model_metadata`

  Metadata for the model. If an error occurs during training or you cancel the training operation, AutoML records the error status in this column. See Model Metadata.

- notes

  Use this column to record notes about the trained model. It also records any error messages that occur during model training.

The Model Object Catalog Table

Models are chunked and stored uncompressed in the model_object_catalog table. Each chunk is saved with the same model_handle.

A call to one of the following routines upgrades the model catalog, and store the model in the model_object_catalog table:

- ML_TRAIN
- ML_MODEL_LOAD
- ML_EXPLAIN
- ML_MODEL_IMPORT
- ML_MODEL_EXPORT

If the call to one of these routines is not successful or is aborted, then the previous model catalog is still available.

The model_object_catalog table has the following columns:

- chunk_id

  A primary key, and an auto-incrementing numeric identifier for the chunk. It is unique for the chunks sharing the same model_handle.

- model_handle

  A primary key, and a foreign key that references model_handle in the MODEL_CATALOG table.

- model_object

  A string in JSON format containing the serialized AutoML model.

See Also

- Review Model Metadata for the Model Catalog Table.
- Review Model Handles and how to retrieve them from the Model Catalog Table.

4.7.1.2 Model Metadata

The model_metadata column in the model catalog allows you to view detailed information on trained models. For example, you can view the algorithm used to train the model, the columns in the training table, and values for the model explanation.

When you run the ML_MODEL_IMPORT routine, the imported table has a model_metadata column that stores the metadata for the table. If you import a model from a table, model_metadata stores the name of the database and table. If you import a model object, model_metadata stores a JSON_OBJECT that contains key-value pairs of the metadata See Section 7.1.4, “ML_MODEL_IMPORT” to learn more.

The default value for model_metadata is NULL.

This topic has the following sections.

- Model Metadata Details
- Query Model Metadata
- See Also

Model Metadata Details

`model_metadata` contains the following metadata as key-value pairs in JSON format:

- `task:` *`string`*

  The task type specified in the `ML_TRAIN` query. The default is `classification` when used with `ML_MODEL_IMPORT`.

- `build_timestamp:` *`number`*

  A timestamp indicating when the model was created (UNIX epoch time). A model is created when the `ML_TRAIN` routine finishes executing.

- `target_column_name:` *`string`*

  The name of the column in the training table that was specified as the target column.

- `train_table_name:` *`string`*

  The name of the input table specified in the `ML_TRAIN` query.

- `column_names:` *`JSON array`*

  The feature columns used to train the model.

- `model_explanation:` *`JSON object literal`*

  The model explanation generated during training. See Generate Model Explanations.

- `notes:` *`string`*

  The `notes` specified in the `ML_TRAIN` query. It also records any error messages that occur during model training.

- `format:` *`string`*

  The model can be in one of the following formats:

  - HWMLv1.0
  - HWMLv2.0
  - ONNXv1.0
  - ONNXv2.0

- `status:` *`string`*

  The status of the model. The default is `Ready` when used with `ML_MODEL_IMPORT`.

  - `Creating`: The model is being created.
  - `Ready`: The model is trained and active.

  - `Error`: Either training was canceled or an error occurred during training. Any error message appears in the `notes` column. The error message also appears in `model_metadata` `notes`.

- `model_quality:` *`string`*

  The quality of the model object for classification and regression tasks. For other tasks, this value is `NULL`. The value is either `low` or `high`.

- `training_time:` *`number`*

  The time in seconds taken to train the model.

- `algorithm_name:` *`string`*

  The name of the chosen algorithm.

- `training_score:` *`number`*

  The cross-validation score achieved for the model by training.

- `n_rows:` *`number`*

  The number of rows in the training table.

- `n_columns:` *`number`*

  The number of columns in the training table.

- `n_selected_rows:` *`number`*

  The number of rows selected by adaptive sampling.

- `n_selected_columns:` *`number`*

  The number of columns selected by feature selection.

- `optimization_metric:` *`string`*

  The optimization metric used for training. See Section 7.1.14, “Optimization and Scoring Metrics” to review available metrics.

- `selected_column_names:` *`JSON array`*

  The names of the columns selected by feature selection.

- `contamination:` *`number`*

  The contamination factor for the anomaly detection task. See Anomaly Detection Options to learn more.

- `options:` *`JSON object literal`*

  The `options` specified in the `ML_TRAIN` query.

- `training_params:` *`JSON object literal`*

  Internal task dependent parameters used during `ML_TRAIN`.

- `onnx_inputs_info:` *`JSON object literal`*

Information about the format of the ONNX model inputs. This only applies to ONNX models. See Manage External ONNX Models.

Do not provide `onnx_inputs_info` if the model is not ONNX format. This generates an error.

- `data_types_map:` *`JSON object literal`*

  This maps the data type of each column to an ONNX model data type. The default value is:

  ```
  JSON_OBJECT("tensor(int64)": "int64", "tensor(float)": "float32", "tensor(string)": "str_")
  ```

- `onnx_outputs_info:` *`JSON object literal`*

  Information about the format of the ONNX model outputs. This only applies to ONNX models. See Manage External ONNX Models.

  Do not provide `onnx_outputs_info` if the model is not ONNX format, or if `task` is `NULL`. This generates an error.

  - `predictions_name:` *`string`*

    This name determines which of the ONNX model outputs is associated with predictions.

  - `prediction_probabilities_name:` *`string`*

    This name determines which of the ONNX model outputs is associated with prediction probabilities.

  - `labels_map:` *`JSON object literal`*

    This maps prediction probabilities to predictions, known as labels.

- `training_drift_metric:` *`JSON object literal`*

  Contains data drift information about the training data. See Analyze Data Drift. This only applies to classification and regression models.

  - `mean:` *`number`*

    The mean value of drift metrics of all the training data. ≥ `0`.

  - `variance:` *`number`*

    The variance value of drift metrics of all the training data. ≥ `0`.

  Both `mean` and `variance` should be low.

- `chunks:` *`number`*

  The total number of chunks that the model has been split into.

Query Model Metadata

You can query the model metadata in the model catalog with the following command. Replace `user1` with your own user name.

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG\G
*************************** 1. row ***************************
JSON_PRETTY(model_metadata): {
  "task": "regression",
```

```
"notes": null,
"chunks": 1,
"format": "HWMLv2.0",
"n_rows": 407284,
"status": "Ready",
"options": {
  "task": "regression",
  "model_explainer": "permutation_importance",
  "prediction_explainer": "permutation_importance"
},
"n_columns": 14,
"column_names": [
  "VendorID",
  "store_and_fwd_flag",
  "RatecodeID",
  "PULocationID",
  "DOLocationID",
  "passenger_count",
  "extra",
  "mta_tax",
  "tolls_amount",
  "improvement_surcharge",
  "trip_type",
  "lpep_pickup_datetime_day",
  "lpep_pickup_datetime_hour",
  "lpep_pickup_datetime_minute"
],
"contamination": null,
"model_quality": "high",
"training_time": 515.13427734375,
"algorithm_name": "RandomForestRegressor",
"training_score": -5.610334873199463,
"build_timestamp": 1730395944,
"n_selected_rows": 130931,
"training_params": {
  "recommend": "ratings",
  "force_use_X": false,
  "recommend_k": 3,
  "remove_seen": true,
  "ranking_topk": 10,
  "lsa_components": 100,
  "ranking_threshold": 1,
  "feedback_threshold": 1
},
"train_table_name": "heatwaveml_bench.nyc_taxi_train",
"model_explanation": {
  "permutation_importance": {
    "extra": 0.0,
    "mta_tax": 0.0019,
    "VendorID": 0.0048,
    "trip_type": 0.0003,
    "RatecodeID": 0.0152,
    "DOLocationID": 0.4178,
    "PULocationID": 0.2714,
    "tolls_amount": 0.0851,
    "passenger_count": 0.0,
    "store_and_fwd_flag": 0.0,
    "improvement_surcharge": 0.0015,
    "lpep_pickup_datetime_day": 0.0,
    "lpep_pickup_datetime_hour": 0.0161,
    "lpep_pickup_datetime_minute": 0.0
  }
},
"n_selected_columns": 9,
"target_column_name": "tip_amount",
"optimization_metric": "neg_mean_squared_error",
"selected_column_names": [
```

```
    "DOLocationID",
    "PULocationID",
    "RatecodeID",
    "VendorID",
    "improvement_surcharge",
    "lpep_pickup_datetime_hour",
    "mta_tax",
    "tolls_amount",
    "trip_type"
  ],
  "training_drift_metric": {
    "mean": 0.3326,
    "variance": 3.2482
  }
}
*************************** 2. row ***************************
JSON_PRETTY(model_metadata): {
  "task": "regression",
  "notes": null,
  "chunks": 0,
  "format": "HWMLv2.0",
  "n_rows": null,
  "status": "Error",
  "options": {},
  "n_columns": null,
  "column_names": null,
  "contamination": null,
  "model_quality": null,
  "training_time": null,
  "algorithm_name": null,
  "training_score": null,
  "build_timestamp": 1730403865,
  "n_selected_rows": null,
  "training_params": null,
  "train_table_name": "nyc_taxi.nyc_taxi_train",
  "model_explanation": {},
  "n_selected_columns": null,
  "target_column_name": "tip_amount",
  "optimization_metric": null,
  "selected_column_names": null,
  "training_drift_metric": {
    "mean": null,
    "variance": null
  }
}
*************************** 3. row ***************************
JSON_PRETTY(model_metadata): {
  "task": "regression",
  "notes": null,
  "chunks": 0,
  "format": "HWMLv2.0",
  "n_rows": null,
  "status": "Creating",
  "options": {},
  "n_columns": null,
  "column_names": null,
  "contamination": null,
  "model_quality": null,
  "training_time": null,
  "algorithm_name": null,
  "training_score": null,
  "build_timestamp": 1730404027,
  "n_selected_rows": null,
  "training_params": null,
  "train_table_name": "nyc_taxi.nyc_taxi_train",
  "model_explanation": {},
  "n_selected_columns": null,
```

```
  "target_column_name": "tip_amount",
  "optimization_metric": null,
  "selected_column_names": null,
  "training_drift_metric": {
    "mean": null,
    "variance": null
  }
}
3 rows in set (0.0859 sec)
```

See Also

- Analyze Data Drift
- Manage External ONNX Models
- The Model Catalog Table
- Generate Model Explanations

4.7.2 Work with Model Handles

When ML_TRAIN trains a model, you have the option to specify a name for the model, which is the model handle. If you do not specify a model handle name, a model handle is automatically generated that is based on the database name, input table name, the user name training the table, and a unique numerical identifier. You must use model handles to run AutoML routines. All model handles must be unique in the model catalog.

This topic has the following sections.

- Before You Begin
- Model Handles Overview
- Query the Model Handle
- Defining Model Handle
- Assign Session Variable to Model Handle
- What's Next

Before You Begin

- Review the The Model Catalog.

Model Handles Overview

If you set the model handle name to a session variable before training a model, the model handle takes that name. Otherwise, a unique model handle is automatically generated. To set your own model name, see Defining Model Handle. The model handle is stored temporarily in a user-defined session variable specified in the ML_TRAIN call. In the following example, @census_model is defined as the model handle session variable with no set model handle name:

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @c
```

While the connection used to run ML_TRAIN remains active, that connection can retrieve the automatically generated model handle by querying the session variable. For example:

```
mysql> SELECT @census_model;
+--------------------------------------------------+
| @census_model                                    |
+--------------------------------------------------+
| census_classification_model                      |
+--------------------------------------------------+
```

Query the Model Handle

Since the session variable for a model handle is only valid for the current session, you can query the model handle name from the model catalog in new sessions.

The following example queries the model handle, the model owner, and the name of the training table from the model catalog table. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+---------------------------------------------------------------+-------------+----------------------
| model_handle                                                  | model_owner | train_table_name
+---------------------------------------------------------------+-------------+----------------------
| census_classification_model                                   | user1       | census_data.census_tra
+---------------------------------------------------------------+-------------+----------------------
18 rows in set (0.0014 sec)
```

Once you have the model handle, you can use it directly in AutoML routines instead of the session variable.

The following example runs `ML_PREDICT_ROW` and uses the model handle.

```
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, 'census_classification_model', NULL);
```

Defining Model Handle

Before training a model, it is good practice to define your own model handle instead of automatically generating one. This allows you to easily remember the model handle for future routines on the trained model instead of having to query it, or depending on the session variable that can no longer be used when the current connection terminates.

To define your own model handle:

1. Set the value of the session variable, which sets the model handle to this same value.

   ```
   mysql> SET @variable = 'model_handle';
   ```

   Replace *`@variable`* and *`model_handle`* with your own definitions. For example:

   ```
   mysql> SET @census_model = 'census_classification_model';
   ```

   When `ML_TRAIN` runs with this session variable, the model handle is set to `census_test`.

   If you set a model handle that already appears in the model catalog, the `ML_TRAIN` routine returns an error.

2. Run the `ML_TRAIN` routine.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @variabl
   ```

   Replace *`table_name`*, *`target_column_name`*, *`task_name`*, and *`variable`* with your own values.

   The following example trains a model with the model handle variable previously set

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task', 'classification'),
```

3. After training the model, query the model catalog to confirm the model handle you defined is there. Replace `user1` with your own user name.

```
mysql> SELECT model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+--------------------------------------------------------------------+-------------+-----------------------
| model_handle                                                       | model_owner | train_table_name
+--------------------------------------------------------------------+-------------+-----------------------
| census_classification_model                                        | user1       | census_data.census_trai
+--------------------------------------------------------------------+-------------+-----------------------
1 row in set (0.0014 sec)
```

Assign Session Variable to Model Handle

If you lose the session variable to a model handle due to a lost connection, you have the option of assigning a new session variable to a model handle in a new connection.

To assign a session variable to a model handle:

1. Set a variable to the model handle. If needed Query the Model Handle.

```
mysql> SET @my_model = 'model_handle';
```

The following example sets the `@my_model` session variable to a model handle.

```
mysql> SET @my_model = 'census_classification_model';
```

2. Confirm the session variable is assigned to the model handle by querying the session variable.

```
mysql> SELECT @my_model;
+--------------------------------------------------------+
| @my_model                                              |
+--------------------------------------------------------+
| census_classification_model                            |
+--------------------------------------------------------+
```

Alternatively, you can assign a session variable to the model handle for the most recently trained model.

1. Set a variable with the query to retrieve the most recent model handle by sorting with the `build_timestamp` parameter in the model catalog. Replace `user1` with your own user name.

```
mysql> SET @variable = (SELECT model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG ORDER BY build_timestamp DES
```

The following example sets the `latest_model` variable.

```
mysql> SET @latest_model = (SELECT model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG ORDER BY timestamp DESC
```

2. Confirm the session variable is assigned to the latest model handle by querying the session variable.

```
mysql> SELECT @latest_model;
+-------------------------+
| @latest_model           |
+-------------------------+
| recommendation_use_case4 |
+-------------------------+
1 row in set (0.0454 sec)
```

What's Next

- Review how to Create a Machine Learning Model.

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.7.3 Unload a Model

The `ML_MODEL_UNLOAD` routine unloads a model from AutoML. Review ML_MODEL_UNLOAD parameter descriptions.

Before You Begin

- Review the following
  - Train a Model
  - Load a Model

Unload a Model

You can verify what models are currently loaded with the `ML_MODEL_ACTIVE` routine before and after unloading the model.

1. Verify what models are currently loaded with the `ML_MODEL_ACTIVE` routine.

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
```

2. Select the session variable created to view all loaded models.

```
mysql> SELECT JSON_PRETTY(@model_info);
+------------------------------------------------------------+
| JSON_PRETTY(@model_info)                                   |
+------------------------------------------------------------+
| [
  {
    "total model size(bytes)": 50209
  },
  {
    "user1": [
      {
        "recommendation_use_case": {
          "format": "HWMLv2.0",
          "model_size(byte)": 15609
        }
      },
      {
        "recommendation_use_case2": {
          "format": "HWMLv2.0",
          "model_size(byte)": 8766
        }
      },
      {
        "recommendation_use_case3": {
          "format": "HWMLv2.0",
          "model_size(byte)": 8402
        }
      },
      {
        "recommendation_use_case4": {
          "format": "HWMLv2.0",
          "model_size(byte)": 17432
        }
      }
    ]
  }
] |
```

```
+-----------------------------------------------------------+
1 row in set (0.0411 sec)
```

3. Refer to the appropriate model handle to unload. Alternatively, use the session variable for the model handle.

   The following example unloads a model by using the model handle:

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD('recommendation_use_case');
   ```

   Where:

   - `recommendation_use_case` is the model handle.

   The following example unloads a model by using the session variable for the model handle:

   ```
   mysql> CALL sys.ML_MODEL_UNLOAD(@recommendation_model);
   ```

   Where:

   - `@recommendation_model` is the assigned session variable for the model handle.

4. Run `ML_MODEL_ACTIVE` again to confirm the model is successfully unloaded

   ```
   mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
   mysql> SELECT JSON_PRETTY(@model_info);
   +-----------------------------------------------------------+
   | JSON_PRETTY(@model_info)                                  |
   +-----------------------------------------------------------+
   | [
     {
       "total model size(bytes)": 34600
     },
     {
       "user1": [
         {
           "recommendation_use_case2": {
             "format": "HWMLv2.0",
             "model_size(byte)": 8766
           }
         },
         {
           "recommendation_use_case3": {
             "format": "HWMLv2.0",
             "model_size(byte)": 8402
           }
         },
         {
           "recommendation_use_case4": {
             "format": "HWMLv2.0",
             "model_size(byte)": 17432
           }
         }
       ]
     }
   ] |
   +-----------------------------------------------------------+
   1 row in set (0.0411 sec)
   ```

   The list of loaded models shows the model is unloaded.

What's Next

- Review how to Create a Machine Learning Model.

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.7.4 View Model Details

To view the details for the models in your model catalog, query the `MODEL_CATALOG` table.

Before You Begin

- Review the following:
  - Create a Machine Learning Model
  - The Model Catalog

View Details for Your Models

The following example queries `model_id`, `model_handle`, and `model_owner`, `train_table_name` from the model catalog. Replace `user1` with your own user name.

```
mysql> SELECT model_id, model_handle, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
+----------+------------------------------------------+-------------+-----------------------------------
| model_id | model_handle                             | model_owner | train_table_name
+----------+------------------------------------------+-------------+-----------------------------------
|        1 | regression_use_case                      | root        | regression_data.house_price_traini
|        2 | forecasting_use_case                     | root        | forecasting_data.electricity_deman
|        3 | anomaly_detection_semi_supervised_use_case | root        | anomaly_data.credit_card_train
|        4 | anomaly_detection_log_use_case           | root        | anomaly_log_data.training_data
|        5 | recommendation_use_case                  | root        | recommendation_data.training_datas
|        6 | topic_modeling_use_case                  | root        | topic_modeling_data.movies
+----------+------------------------------------------+-------------+-----------------------------------
```

Where:

- `model_id` is a unique numeric identifier for the model.
- `model_owner` is the user that created the model.
- `model_handle` is the handle by which the model is called.
- `ML_SCHEMA_user1.MODEL_CATALOG` is the fully qualified name of the `MODEL_CATALOG` table. The schema is named for the owning user.

The output displays details from only a few `MODEL_CATALOG` table columns. For other columns you can query, see The Model Catalog.

View Model Explanations

The `ML_EXPLAIN` routine generates model explanations and stores them in the model catalog. See Generate Model Explanations to learn more.

A model explanation helps you identify the features that are most important to the model overall. Feature importance is presented as an attribution value. A positive value indicates that a feature contributed toward the prediction. A negative value can have different interpretations depending on the specific model explainer used for the model. For example, a negative value for the permutation importance explainer means that the feature is not important.

To view a model explanation, you can query the `model_explanation` column from the model catalog by referencing the model handle. Review how to Query the Model Handle.

```
mysql> SELECT column FROM ML_SCHEMA_user name.MODEL_CATALOG where model_handle='model_handle';
```

The following example queries one of the model handles and views the model explanation for that model. Optionally, use JSON_PRETTY to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG where model_handle='census_mod
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0305,
    "sex": 0.0023,
    "race": 0.0017,
    "fnlwgt": 0.0025,
    "education": 0.0013,
    "workclass": 0.0043,
    "occupation": 0.0229,
    "capital-gain": 0.0495,
    "capital-loss": 0.0156,
    "relationship": 0.0267,
    "education-num": 0.0371,
    "hours-per-week": 0.0142,
    "marital-status": 0.0267,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0447 sec)
```

Where:

- ML_SCHEMA_*user1*.MODEL_CATALOG is the fully qualified name of the MODEL_CATALOG table. The schema is named for the user that created the model.
- census_data.census_train_user1_1744548610842 is the model handle. See Work with Model Handles.

The output displays feature importance values for each column by using the permutation_importance model explainer.

Alternatively, you can query the model explanation by using the valid session variable for the model handle. Optionally, use JSON_PRETTY to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_admin.MODEL_CATALOG where model_handle=@census_mod
+-----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+-----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0305,
    "sex": 0.0023,
    "race": 0.0017,
    "fnlwgt": 0.0025,
    "education": 0.0013,
    "workclass": 0.0043,
    "occupation": 0.0229,
    "capital-gain": 0.0495,
    "capital-loss": 0.0156,
    "relationship": 0.0267,
    "education-num": 0.0371,
    "hours-per-week": 0.0142,
    "marital-status": 0.0267,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
```

```
1 row in set (0.0447 sec)
```

See Work with Model Handles to learn more.

What's Next

- Review the The Model Catalog.
- Review how to Work with Model Handles.

4.7.5 Delete a Model

Users that create models or have the required privileges to a model on the `MODEL_CATALOG` table can delete them.

Before You Begin

- Review how to Create a Machine Learning Model.
- Review how to Share a Model.

Delete a Model

To delete a model from the model catalog table:

1. Query the model catalog table for the `model_id`, `model_owner`, and `train_table_name`. Identify the `model_id` for model you want to delete. Replace `user1` with your own user name.

   ```
   mysql> SELECT model_id, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
   +----------+-------------+-----------------------------------------------+
   | model_id | model_owner | train_table_name                              |
   +----------+-------------+-----------------------------------------------+
   |        1 | user1       | ml_benchmark.sentiment_model_creation         |
   |        2 | user1       | ml_data.iris_train                            |
   |        3 | user1       | census_data.census_train                      |
   +----------+-------------+-----------------------------------------------+
   3 rows in set (0.0008 sec)
   ```

   The requested columns from the model catalog table display.

   In this case, the model with `model_id` 3 is deleted.

2. Delete the model from the model catalog table.

   ```
   mysql> DELETE FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_id = 3;
   ```

   Where:

   - `ML_SCHEMA_user1.MODEL_CATALOG` is the fully qualified name of the `MODEL_CATALOG` table. The schema is named for the user that created the model.
   - `model_id = 3` is the ID of the model you want to delete.

3. Confirm the model is removed from the model catalog table. Replace *user1* with your own user name.

   ```
   mysql> SELECT model_id, model_owner, train_table_name FROM ML_SCHEMA_user1.MODEL_CATALOG;
   +----------+-------------+-----------------------------------------------+
   | model_id | model_owner | train_table_name                              |
   +----------+-------------+-----------------------------------------------+
   |        1 | user1       | ml_benchmark.sentiment_model_creation         |
   |        2 | user1       | ml_data.iris_train                            |
   ```

```
+----------+------------+---------------------------------------------+
2 rows in set (0.0008 sec)
```

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.7.6 Share a Model

This topic describes how to grant other users access to a model you create.

This topic has the following sections.

- Before You Begin
- Share Your Models
- Export the Model to Share
- Set Up Other User with Required Privileges
- Importing Shared Model
- Run AutoML Routines on Imported Model
- What's Next

Before You Begin

- Review AutoML Privileges.

Share Your Models

To share a model you created, you can use the `ML_MODEL_EXPORT` and `ML_MODEL_IMPORT` routines. `ML_MODEL_EXPORT` exports the model to share to a user-defined table that both users need the required privileges to access. `ML_MODEL_IMPORT` imports the model to the user's model catalog. The other user can then run AutoML commands on the imported model.

In the following tasks, the `admin` user gives access to their model to the `user1` user. The trained table, `bank_train`, is in the `bank_marketing` database.

Export the Model to Share

The `admin` user needs to export the model to share to a user-defined table that both users can access. In this use case, the user exports the model to their own model catalog.

1. As the `admin` user, train and load the model to export. See Train a Model and Load a Model.

2. Export the model to a table in the model catalog. Use the assigned session variable for the model handle. If you need to query the model handle, see Work with Model Handles.

   ```
   mysql> CALL sys.ML_MODEL_EXPORT (model_handle, output_table_name);
   ```

   Replace *`model_handle`* and *`output_table_name`* with your own values. For example:

   ```
   mysql> CALL sys.ML_MODEL_EXPORT(@bank_model, 'ML_SCHEMA_admin.model_export');
   ```

   Where:

   - `@bank_model` is the assigned session variable for the model handle of the trained model.

- ML_SCHEMA_admin.model_export is the fully qualified name of the table that contains the training dataset (*schema_name.table_name*).

3. Run the SHOW CREATE TABLE command to confirm the table was created with the recommended parameters for importing. See ML_MODEL_IMPORT to learn more.

```
mysql> SHOW CREATE TABLE ML_SCHEMA_admin.model_export;
+--------------+--------------------------------------------------------------------------------
| Table        | Create Table
+--------------+--------------------------------------------------------------------------------
| model_export | CREATE TABLE `model_export` (
  `chunk_id` int NOT NULL AUTO_INCREMENT,
  `model_object` longtext,
  `model_metadata` json DEFAULT NULL,
  PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+--------------------------------------------------------------------------------
1 row in set (0.0527 sec)
```

Set Up Other User with Required Privileges

The admin user needs to grant the required privileges to user1, so that user can access exported model and import it into their own model catalog.

1. If not done already, create the other user account (user1). See CREATE USER Statement to learn more.

2. Run these commands to grant the required privileges to the other user, so they can access the following:

   - AutoML routines on the MySQL sys schema.
   - The model catalog for both users.
   - The database with the trained model.

   See AutoML Privileges to learn more.

```
mysql> GRANT SELECT, EXECUTE ON sys.* TO 'user1'@'%';
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON ML_SCHEMA_user1.* TO 'user1'@
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON ML_SCHEMA_admin.* TO 'user1'@
mysql> GRANT SELECT, ALTER, INSERT, CREATE, UPDATE, DROP, GRANT OPTION ON bank_marketing.* TO 'user1'@'
mysql> GRANT SELECT ON performance_schema.rpd_tables TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_table_id TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_query_stats TO 'user1'@'%';
mysql> GRANT SELECT ON performance_schema.rpd_ml_stats TO 'user1'@'%';
```

   Where:

   - ML_SCHEMA_user1.* and ML_SCHEMA_user1.* gives access to the model catalog for both users.
   - bank_marketing is the database that contains the trained table.

Importing Shared Model

The user1 user can now import the exported model to their own model catalog.

1. Log in to the DB system as the other user (user1).

2. Import the model the admin user previously exported into the model catalog for user1.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

Replace *model_object*, *model_metadata*, and *model_handle* with your own values. For example:

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'ML_SCHEMA_admin', 'table', 'model_export'), @b
```

- `NULL` means that a model from a table is imported, and not a model object.
- `JSON_OBJECT` sets key-value pairs for the database and table of the exported table to import.
- `@bank_export` is the assigned session variable for the imported model handle.

3. Load the imported model. Use the assigned session variable set for the imported model handle in the previous command.

```
mysql> CALL sys.ML_MODEL_LOAD(@bank_export, NULL);
```

4. Optionally, query `model_object` and `model_object_size` from the model catalog for the loaded model to confirm the model imported successfully.

```
mysql> SELECT model_object, model_object_size FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@bank_e
+--------------+-------------------+
| model_object | model_object_size |
+--------------+-------------------+
| NULL         |            331860 |
+--------------+-------------------+
1 row in set (0.0478 sec)
```

Confirm the `model_object_size` is not 0.

5. Optionally, query `chunk_id` and `LENGTH(model_object)` from the model object catalog for the loaded model to confirm the model imported successfully.

```
mysql> SELECT chunk_id, LENGTH(model_object) FROM ML_SCHEMA_user1.model_object_catalog WHERE model_handle=@
+----------+----------------------+
| chunk_id | LENGTH(model_object) |
+----------+----------------------+
|        1 |               331860 |
+----------+----------------------+
1 row in set (0.0465 sec)
```

Confirm the `chunk_id` value is 1 and `LENGTH(model_object)` is not 0.

Run AutoML Routines on Imported Model

Confirm the `user1` user can run AutoML commands. The following example generates a table of predictions for the imported model.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *table_name*, *model_handle*, *output_table_name*), and *options* with your own values. For example:

```
mysql> CALL sys.ML_PREDICT_TABLE('bank_marketing.bank_train', @bank_export, 'bank_marketing.bank_predictions',
```

Where:

- `bank_marketing.bank_train` is the fully qualified name of the table that contains the training dataset (*schema_name.table_name*).
- `@bank_export` is the assigned session variable for the imported model handle.

- `bank_marketing.bank_predictions` is the fully qualified name of the output table that contains the predictions (*`schema_name.table_name`*).

Optionally, use the database with the output table and query a sample.

```
mysql> USE bank_marketing;
mysql> SELECT * FROM bank_predictions limit 5;
+-------------------+-----+-------------+---------+-----------+---------+---------+---------+------+-----
| _4aad19ca6e_pk_id | age | job         | marital | education | default | balance | housing | loan | conta
+-------------------+-----+-------------+---------+-----------+---------+---------+---------+------+-----
|                 1 |  30 | management  | single  | tertiary  | no      |     149 | yes     | no   | unkno
|                 2 |  46 | blue-collar | married | secondary | no      |   -1400 | yes     | no   | telep
|                 3 |  33 | entrepreneur | married | secondary | no     |    -118 | yes     | yes  | unkno
|                 4 |  43 | blue-collar | married | secondary | no      |    2160 | no      | no   | cellu
|                 5 |  38 | management  | married | tertiary  | no      |    3452 | no      | no   | cellu
+-------------------+-----+-------------+---------+-----------+---------+---------+---------+------+-----
5 rows in set (0.0425 sec)
```

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.7.7 Manage External ONNX Models

AutoML supports the upload of pre-trained models in ONNX (Open Neural Network Exchange) format to the model catalog. Load them with the `ML_MODEL_IMPORT` routine. After import, you can use AutoML routines with ONNX models.

4.7.7.1 ONNX Models Overview

You cannot directly load models in ONNX format (`.onnx`) into a MySQL table. The models require string serialization and conversion to Base64 encoding before you use the `ML_MODEL_IMPORT` routine.

AutoML supports the following ONNX model types:

- An ONNX model that has only one input, and it is the entire MySQL table.
- An ONNX model that has more than one input, and each input is one column in the MySQL table.

For example, AutoML does not support an ONNX model that takes more than one input, and each input is associated with more than one column in the MySQL table.

The first dimension of the input to the ONNX model provided by the ONNX model `get_inputs()` API should be the batch size. This should be `None`, a string, or an integer. `None` or string indicate a variable batch size, and an integer indicates a fixed batch size.

Examples of input shapes:

```
[None, 2]

['batch_size', 2, 3]

[1, 14]
```

All other dimensions should be integers. For example, AutoML does not support an input shape similar to the following:

```
input shape = ['batch_size', 'sequence_length']
```

The output of an ONNX model is a list of results. The ONNX API documentation defines the results as a numpy array, a list, a dictionary, or a sparse tensor. AutoML only supports a numpy array, a list, and a dictionary.

- Numpy array examples:

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor'], dtype=object)

array([0, 2, 0, 0], dtype=int64)

array([[0.8896357 , 0.11036429],
       [0.28360802, 0.716392  ],
       [0.9404001 , 0.05959991],
       [0.5655978 , 0.43440223]], dtype=float32)

array([[0.96875435],
       [1.081366  ],
       [0.5736201 ],
       [0.90711355]], dtype=float32)
```

- Simple list examples:

```
['Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor']

[0, 2, 0, 0]
```

- List of lists examples:

```
[[0.8896357 , 0.110364],
[0.28360802, 0.716392],
[0.9404001 , 0.059599],
[0.5655978 , 0.434402]]

[[[0.8896357] , [0.110364]],
[[0.28360802], [0.716392]],
[[0.9404001] , [0.059599]],
[[0.5655978] , [0.434402]]]

[[0.968754],
[1.081366],
[0.573620],
[0.907113]]

[[[0.968754]],
[[1.081366]],
[[0.573620]],
[[0.907113]]]
```

- Dictionary examples:

```
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.0, 'Iris-virginica': 0.999}

{0: 0.1, 1: 0.9}
```

- List of dictionaries examples:

```
[{'Iris-setosa': 0.0, 'Iris-versicolor': 0.0, 'Iris-virginica': 0.999},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.999, 'Iris-virginica': 0.0},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.589, 'Iris-virginica': 0.409},
{'Iris-setosa': 0.0, 'Iris-versicolor': 0.809, 'Iris-virginica': 0.190}]

[{0: 1.0, 1: 0.0, 2: 0.0},
{0: 0.0, 1: 0.0, 2: 1.0},
{0: 1.0, 1: 0.0, 2: 0.0},
{0: 1.0, 1: 0.0, 2: 0.0}]

[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]
```

```
[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]

[{0: 0.176, 1: 0.823},
{0: 0.176, 1: 0.823},
{0: 0.264, 1: 0.735},
{0: 0.875, 1: 0.124}]
```

For classification and regression tasks, AutoML only supports model explainers and scoring for variable batch sizes.

For forecasting, anomaly detection and recommendation tasks, AutoML does not support model explainers and scoring. The prediction column must contain a JSON object literal of name value keys. For example, for three outputs:

```
{output1: value1, output2: value2, output3: value3}
```

What's Next

- Learn about ONNX Model Metadata.

4.7.7.2 ONNX Model Metadata

To learn more about model metadata in the model catalog, see Model Metadata. The model metadata includes `onnx_inputs_info` and `onnx_outputs_info`.

- `onnx_inputs_info` includes `data_types_map`. See Model Metadata for the default value.
- `onnx_outputs_info` includes `predictions_name`, `prediction_probabilities_name`, and `labels_map`.

ONNX Inputs Info

Use the `data_types_map` to map the data type of each column to an ONNX model data type. For example, to convert inputs of the type `tensor(float)` to `float64`:

```
data_types_map = {"tensor(float)": "float64"}
```

AutoML first checks the user `data_types_map`, and then the default `data_types_map` to check if the data type exists. AutoML supports the following numpy data types:

Table 4.1 Supported numpy data types

str_	unicode_	int8	int16	int32	int64	int_	uint16
uint32	uint64	byte	ubyte	short	ushort	intc	uintc
uint	longlong	ulonglong	intp	uintp	float16	float32	float64
half	single	longfloat	double	longdouble	bool_	datetime64	complex_
complex64	complex128	complex256	csingle	cdouble	clongdouble		

The use of any other numpy data type causes an error.

ONNX Outputs Info

Use `predictions_name` to determine which of the ONNX model outputs is associated with predictions. Use `prediction_probabilities_name` to determine which of the ONNX model outputs is associated with prediction probabilities. Use use a `labels_map` to map prediction probabilities to predictions, known as labels.

For regression tasks:

- If the ONNX model generates only one output, then predictions_name is optional.
- If the ONNX model generates more than one output, then predictions_name is required.
- Do not provide prediction_probabilities_name as this causes an error.

For classification tasks:

- Use predictions_name, prediction_probabilities_name, or both. Failure to provide at least one causes an error.
- The model explainers SHAP, Fast SHAP, and Partial Dependence require prediction_probabilities_name.

Only use a labels_map with classification tasks. A labels_map requires predictions_probabilities_name. The use of a labels_map with any other task, or with predictions_name or without predictions_probabilities_name causes an error.

If the task is NULL, do not provide predictions_name or prediction_probabilities_name as this causes an error.

An example of a predictions_probabilities_name with a labels_map produces these labels:

```
predictions_probabilities_name = array([[0.35, 0.50, 0.15],
                                        [0.10, 0.20, 0.70],
                                        [0.90, 0.05, 0.05],
                                        [0.55, 0.05, 0.40]], dtype=float32)

labels_map = {0:'Iris-virginica', 1:'Iris-versicolor', 2:'Iris-setosa'}

labels=['Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica']
```

AutoML adds a note for ONNX models that have inputs with four dimensions about the reshaping of data to a suitable shape for an ONNX model. This would typically be for ONNX models that are trained on image data.

An example of this note added to the ml_results column:

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus_v5.mnist_test_temp', @model,
          'mlcorpus_v5.`mnist_predictions`', NULL);
Query OK, 0 rows affected (20.6296 sec)

mysql> SELECT ml_results FROM mnist_predictions;;
+-----------------------------------------------------------------------------------------------------------
| ml_results
+-----------------------------------------------------------------------------------------------------------
| {'predictions': {'prediction': 7}, 'Notes': 'Input data is reshaped into (1, 28, 28).', 'probabilities': {0:
+-----------------------------------------------------------------------------------------------------------
```

See Also

- Review The Model Catalog.

4.7.7.3 Importing an External ONNX Model

This topic describes how to import an external ONNX model.

This topic has the following sections. Refer to the steps to import an ONNX model. There are also examples for your reference.

Before You Begin

- Review the following:
  - ONNX Models Overview
  - ONNX Model Metadata
- Review ONNX Model Metadata.

Ways to Import External ONNX Model

You have the following ways to import an external ONNX model.

- Import model as a string: For smaller models, you can copy the encoded string and paste it into a session variable or temporary table column. You can then import the table with the copied string. To do this, you run the `ML_MODEL_IMPORT` routine and import the model as a pre-processed model object.
- Import model directly from a table: For larger models, you can load the entire file into a table with the appropriate parameters. You can then import the table directly into your model catalog. If needed, you can load the model in batches of smaller files. To do this, you run the `ML_MODEL_IMPORT` routine and import the model as a table.

  The table that you load the model into must have the following columns:

  - `chunk_id`: The recommended parameters are `INT AUTO_INCREMENT PRIMARY KEY`. There must be only one row in the table with `chunk_id` = 1.
  - `model_object`: The recommended parameters are `LONGTEXT NOT NULL`.
  - `model_metadata`: The recommended parameters are `JSON DEFAULT NULL`.

Workflow to Import an ONNX Model

The workflow to import an ONNX model includes the following:

1. Convert the ONNX file to Base 64 encoding and carry out sting serialization. See Encoding ONNX File.

2. Depending on the size of the model, select if you want to import the model as a string in a pre-processed model object (smaller files) or as a table (larger files). Then, refer to the appropriate section to prepare the model file. See either Preparing to Import ONNX Model as a Pre-Processed Object or Preparing to Import ONNX Model as a Table.

3. Define the model metadata as needed depending on the type of machine learning task for the model. See Defining Model Metadata.

4. Import the model by using the `ML_MODEL_IMPORT` routine. See either Importing ONNX Model as a Pre-processed Object to import the model as a string or Importing ONNX Model as a Table.

Encoding ONNX File

Before importing an ONNX model, you must convert the ONNX file to Base 64 encoding and carry out string serialization. Do this with the Python base64 module. Ensure you have the appropriate version of Python installed.

To encode the ONNX file:

1. Open a terminal window (command prompt on Windows).

2. Install the ONNX library.

```
pip install onnx
```

3. Launch Python and run the following code.

```
# python3 encode_onnx_base64.py
import onnx
import base64

with open("output_file_name", "wb") as f:
    model = onnx.load("input_file_name")
    f.write(base64.b64encode(model.SerializeToString()))
```

Replace `input_file_name` with the full file path to the ONNX file and `output_file_name` with the desired file name for the encoded file. If needed, set a file path for the output file.

The following example converts the `/Users/user1/iris.onnx` file and creates the output file `iris_base64.onnx`.

```
# python3 encode_onnx_base64.py
import onnx
import base64

with open("iris_base64.onnx", "wb") as f:
    model = onnx.load("/Users/user1/iris.onnx")
    f.write(base64.b64encode(model.SerializeToString()))
```

After encoding the ONNX file, select the method to import the model and review the appropriate steps.

- Preparing to Import ONNX Model as a Pre-Processed Object
- Preparing to Import ONNX Model as a Table

Preparing to Import ONNX Model as a Pre-Processed Object

For smaller model files, you can import the ONNX model as a string into a pre-processed object.

To prepare to import the ONNX Model as a string:

1. Open the encoded file and copy the string.

2. Connect to the MySQL server.

3. Copy and paste the converted string for the file into a session variable. For example:

```
mysql> SET @onnx_string_model_object='ONNX_file_string';
```

Alternatively, you can load the encoded file directly into a table column. Make sure you do the following:

- Set the appropriate `local-infile` setting for the client. The server setting `local_infile=ON` is enabled by default. Verify with your admin before using these settings. See Security Considerations for LOAD DATA LOCAL to learn more.
- Upload the file to the appropriate folder in the MySQL server based on the `secure_file_priv` setting. To review this setting, connect to the MySQL server and run the following command:

```
mysql> SHOW VARIABLES LIKE 'secure_file_priv';
```

To load the encoded file directly into a table column:

1. From a terminal window, upload the ONNX file to the folder of your username in the compute instance.

   ```
   $> scp -v -i ssh-key.key /Users/user1/iris_base64.onnx user1@ComputeInstancePublicIP:/home/user1/
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`/Users/user1/iris_base64.onnx`*: The full file path to the ONNX file on your device.
   - *`user1@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
   - *`/home/user1/`*: The appropriate file path to your username in the compute instance.

2. Once the upload successfully completes, SSH into the compute instance.

   ```
   $> ssh -i ssh-key.key user1@computeInstancePublicIP
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`user1@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.

3. Change the directory to the one for your username.

   ```
   $> cd /home/user1
   ```

   Replace `user1` with your own username.

4. Create a copy of the ONNX file.

   ```
   $> touch iris_base64.onnx
   ```

   Replace `iris_base64.onnx` with the file name of the ONNX file.

5. Copy the ONNX file to the appropriate folder in the MySQL server based on the `secure_file_priv` setting.

   ```
   $> sudo cp iris_base64.onnx /var/lib/mysql-files
   ```

Replace the following:

- *iris_base64.onnx*: The file name of the ONNX file.
- */var/lib/mysql-files*: The file path based on the secure_file_priv setting.

6. Update the owner and group of the file path previously specified that has the uploaded ONNX file.

```
$> sudo chown -R mysql:mysql /var/lib/mysql-files
```

   Replace */var/lib/mysql-files* with the file path previously specified.

7. Connect to the MySQL server with the local-infile setting to 1.

```
> mysql -u user1 -p --local-infile=1
```

   Replace user1 with your MySQL username.

8. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

9. Create a table with only one column to store the string.

   The following example creates the onnx_temp table with the onnx_string column with the LONGTEXT data type.

```
mysql> CREATE TABLE onnx_temp (onnx_string LONGTEXT);
```

10. Use a LOAD DATA INFILE statement to load the pre-processed .onnx file into the temporary table.

   The following example loads the iris_base64.onnx file with the string into the onnx_string column in the onnx_temp table.

```
mysql> LOAD DATA INFILE 'iris_base64.onnx'
       INTO TABLE onnx_temp
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r' (onnx_string);
```

11. Insert the loaded string into a session variable.

   The following example loads the loaded string in the onnx_string column into the @onnx_table_model_object session variable.

```
mysql> SELECT onnx_string FROM onnx_temp INTO @onnx_table_model_object;
```

After preparing the model, you can Defining Model Metadata.

Preparing to Import ONNX Model as a Table

For larger model files, you must import the model as a table. Make sure you do the following:

- Set the appropriate local-infile setting for the client. The server setting local_infile=ON is enabled by default. Verify with your admin before using these settings. See Security Considerations for LOAD DATA LOCAL to learn more.
- Upload the file to the appropriate folder in the MySQL server based on the secure_file_priv setting. To review this setting, connect to the MySQL server and run the following command:

```
mysql> SHOW VARIABLES LIKE 'secure_file_priv';
```

To import the model as a table:

1. From a terminal window, upload the ONNX file to the folder of your username in the compute instance.

   ```
   $> scp -v -i ssh-key.key /Users/user1/iris_base64.onnx user1@ComputeInstancePublicIP:/home/user1/
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`/Users/user1/iris_base64.onnx`*: The full file path to the ONNX file on your device.
   - *`user1@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.
   - *`/home/user1/`*: The appropriate file path to your username in the compute instance.

2. Once the upload successfully completes, SSH into the compute instance.

   ```
   $> ssh -i ssh-key.key user1@computeInstancePublicIP
   ```

   Replace the following:

   - *`ssh-key.key`*: The full file path to the SSH key file (.key) for the compute instance.
   - *`user1@ComputeInstancePublicIP`*: The appropriate username and public IP for the compute instance.

3. Change the directory to the one for your username.

   ```
   $> cd /home/user1
   ```

   Replace `user1` with your own username.

4. Create a copy of the ONNX file.

   ```
   $> touch iris_base64.onnx
   ```

   Replace `iris_base64.onnx` with the file name of the ONNX file.

5. Copy the ONNX file to the appropriate folder in the MySQL server based on the `secure_file_priv` setting.

   ```
   $> sudo cp iris_base64.onnx /var/lib/mysql-files
   ```

   Replace the following:

   - *`iris_base64.onnx`*: The file name of the ONNX file.
   - *`/var/lib/mysql-files`*: The file path based on the `secure_file_priv` setting.

6. Update the owner and group of the file path previously specified that has the uploaded ONNX file.

   ```
   $> sudo chown -R mysql:mysql /var/lib/mysql-files
   ```

   Replace *`/var/lib/mysql-files`* with the file path previously specified.

7. Connect to the MySQL server with the `local-infile` setting to `1`.

```
> mysql -u user1 -p --local-infile=1
```

Replace user1 with your MySQL username.

8. Create and use the database to store the table. For example:

```
mysql> CREATE DATABASE onnx_model;
mysql> USE onnx_model;
```

9. Create a table to store the model. The table must have the three required columns to store the details for the model (chunk_id, model_object, and model_metadata). See ML_MODEL_IMPORT Overview. For example:

```
mysql> CREATE TABLE model_table (chunk_id INT AUTO_INCREMENT PRIMARY KEY, model_object LONGTEXT NOT NULL, m
```

10. Use a LOAD DATA INFILE statement to load the model. If needed, load the model in batches of files depending on the size of the model. See LOAD DATA Statement to learn more. The following example loads the model in three separate files into the model_object column in the model_table table previously created:

```
mysql> LOAD DATA INFILE '/onnx_examples/x00'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (34.96 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x01'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (32.74 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x02'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (11.90 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

After preparing the model, you can Defining Model Metadata.

Defining Model Metadata

After preparing the ONNX model (either as a string or table), define the metadata for the model as required. See Model Metadata and ONNX Model Metadata to learn more about requirements depending on the task type of the model.

To define the metadata for the ONNX model:

1. If including the column names for the model in the metadata, you have the option to set them into a JSON object as key-value pairs.

```
mysql> SET @variable = JSON_OBJECT("key","value"[,"key","value"] ...);
```

For example:

```
mysql> SET @column_names = JSON_OBJECT("0","f1", "1","f2", "2","f3");
```

2. Set the metadata for the model as required into a JSON object as key-value pairs. To learn more about metadata requirements, see ONNX Model Metadata. You can also include additional information that allows you to properly configure input tables and columns for generating predictions.

```
mysql> SET @variable = JSON_OBJECT("key","value"[,"key","value"] ...);
```

The following example shows how to define the metadata if you import the model as a string (pre-processed object). The `predictions_name` and `prediction_probabilities_name` variables are provided because it is a classification task. Including the `column_names` allows you to refer to the metadata to ensure that input tables for predictions have the same details. Otherwise an error generates.

```
mysql> SET @model_metadata = JSON_OBJECT('task','classification','onnx_outputs_info',
       JSON_OBJECT('predictions_name','label','prediction_probabilities_name', 'probabilities'),'target
       'train_table_name','mlcorpus.`classification_3_table`','column_names',@column_names,'notes','use
       'training_score',0.734,'training_time',100.34,'n_rows',1000,'n_columns',3,'algorithm_name','xgbo
```

The following example shows how to define the metadata if you import the model from a table. The `predictions_name` and `prediction_probabilities_name` variables are provided because it is a classification task. After defining the metadata, update the metadata for the temporary table for the row that is `chunk_id=1`.

```
mysql> SET @model_metadata = JSON_OBJECT('task','classification','onnx_outputs_info',
       JSON_OBJECT('predictions_name','label','prediction_probabilities_name', 'probabilities'),'target

mysql> UPDATE mlcorpus.model_table SET model_metadata=@model_metadata WHERE chunk_id=1;
```

Depending on how you prepared the model, follow the appropriate steps to import the model:

- Importing ONNX Model as a Pre-processed Object
- Importing ONNX Model as a Table

Importing ONNX Model as a Pre-processed Object

If you followed the steps to Preparing to Import ONNX Model as a Pre-Processed Object, review the following steps to import the model as a pre-processed object.

To import the model as a pre-processed object:

1. Optionally, define the model handle for the imported model instead of automatically generating one. See Work with Model Handles.

```
mysql> SET @variable = 'model_handle';
```

For example:

```
mysql> SET @model = 'onnx_model_string';
```

2. Run `ML_MODEL_IMPORT` to import the model.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

Since you are importing a pre-processed object, the `model_object` is defined by the string you previously set in the in either the `@onnx_string_model_object` or `@onnx_table_model_object` session variable. The `model_metadata` is defined by the metadata previously set in the `@model_metadata` session variable. The `model_handle` is defined by the session variable created for the model handle.

See the following example:

```
mysql> CALL sys.ML_MODEL_IMPORT(@onnx_string_model_object, @model_metadata, @model);
```

3. Confirm the model successfully loaded by querying the `model_id` and `model_handle` from the model catalog. Query the model by using the model handle previously created. Replace `user1` with your own MySQL user name.

```
mysql> SELECT model_id, model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle='onnx_model_stri
+----------+-------------------+
| model_id | model_handle      |
+----------+-------------------+
|       1  | onnx_model_table  |
+----------+-------------------+
1 row in set (0.0485 sec)
```

4. To load the model into MySQL AI so you can start using it with MySQL AI routines, run `ML_MODEL_LOAD`.

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, NULL);
```

For example:

```
mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
```

Importing ONNX Model as a Table

If you followed the steps to Preparing to Import ONNX Model as a Table, review the following steps to import the model as a table.

To import the model as a table:

1. Optionally, define the model handle for the imported model instead of automatically generating one. See Work with Model Handles.

```
mysql> SET @variable = 'model_handle';
```

For example:

```
mysql> SET @model = 'onnx_model_table';
```

2. Run `ML_MODEL_IMPORT` to import the model.

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);
```

Since you are importing a table, the `model_object` is set to `NULL`. The `model_metadata` is defined by the schema name and table name storing the string for the ONNX model. The metadata for the model is stored in the table when following the steps to Defining Model Metadata. The `model_handle` is defined by the session variable created for the model handle.

See the following example:

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'onnx_models', 'table', 'model_table'), @model)
```

3. Confirm the model successfully loaded by querying the `model_id` and `model_handle` from the model catalog. Query the model by using the model handle previously created. Replace `user1` with your own MySQL user name.

```
mysql> SELECT model_id, model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle='onnx_model_
+----------+--------------------+
| model_id | model_handle       |
+----------+--------------------+
|       2  | onnx_model_table   |
+----------+--------------------+
1 row in set (0.0485 sec)
```

4. To load the model into MySQL AI so you can start using it with MySQL AI routines, run `ML_MODEL_LOAD`.

   ```
   mysql> CALL sys.ML_MODEL_LOAD(model_handle, NULL);
   ```

   For example:

   ```
   mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
   ```

ONNX Import Examples

Review the following additional examples for importing ONNX models.

- In the following example, a ONNX model for classification is imported. Then, the model is used to generate predictions, a score, and prediction explainers for a dataset in MySQL AI.

```
mysql> SET @model = 'sklearn_pipeline_classification_3_onnx';
Query OK, 0 rows affected (0.0003 sec)

mysql> SET @model_metadata = JSON_OBJECT('task','classification', 'onnx_outputs_info',
       JSON_OBJECT('predictions_name','label','prediction_probabilities_name', 'probabilities'));
Query OK, 0 rows affected (0.0003 sec)

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_sklearn_pipeline_classification_3, @model_metadata, @model)
Query OK, 0 rows affected (1.2438 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5372 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.classification_3_predict', @model, 'mlcorpus.predictions', NU
Query OK, 0 rows affected (0.8743 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+-------------------+----+----+-----+------------+-------------------------------------------------------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | ml_results
+-------------------+----+----+-----+------------+-------------------------------------------------------
|                 1 | a  | 20 | 1.2 |          0 | {"predictions": {"prediction": 0}, "probabilities": {
|                 2 | b  | 21 | 3.6 |          1 | {"predictions": {"prediction": 1}, "probabilities": {
|                 3 | c  | 19 | 7.8 |          1 | {"predictions": {"prediction": 1}, "probabilities": {
|                 4 | d  | 18 |   9 |          0 | {"predictions": {"prediction": 0}, "probabilities": {
|                 5 | e  | 17 | 3.6 |          1 | {"predictions": {"prediction": 1}, "probabilities": {
+-------------------+----+----+-----+------------+-------------------------------------------------------
5 rows in set (0.0005 sec)

mysql> CALL sys.ML_SCORE('mlcorpus.classification_3_table','target', @model, 'accuracy', @score, NULL);
Query OK, 0 rows affected (0.9573 sec)

mysql> SELECT @score;
+--------+
| @score |
+--------+
|      1 |
+--------+
1 row in set (0.0003 sec)

mysql> CALL sys.ML_EXPLAIN('mlcorpus.classification_3_table', 'target', @model, JSON_OBJECT('model_expla
Query OK, 0 rows affected (10.1771 sec)
```

```
mysql> SELECT model_explanation FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------------------------------------------------+
| model_explanation                                    |
+------------------------------------------------------+
| {"shap": {"f1": 0.0928, "f2": 0.0007, "f3": 0.0039}} |
+------------------------------------------------------+
1 row in set (0.0005 sec)

mysql> CALL sys.ML_EXPLAIN_TABLE('mlcorpus.classification_3_predict', @model, 'mlcorpus.explanations_shap',
Query OK, 0 rows affected (7.6577 sec)

mysql> SELECT * FROM mlcorpus.explanations_shap;
+-------------------+----+----+-----+------------+----------------+-----------------+----------------+------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | f1_attribution | f2_attribution  | f3_attribution | ml_re
+-------------------+----+----+-----+------------+----------------+-----------------+----------------+------
|                 1 | a  | 20 | 1.2 |          0 |       0.116909 |     0.000591494 |    -0.00524929 | {"pre
|                 2 | b  | 21 | 3.6 |          1 |      0.0772133 |     -0.00110559 |     0.00219658 | {"pre
|                 3 | c  | 19 | 7.8 |          1 |      0.0781372 | 0.0000000913938 |    -0.00324671 | {"pre
|                 4 | d  | 18 |   9 |          0 |       0.115209 |    -0.000592354 |     0.00639341 | {"pre
|                 5 | e  | 17 | 3.6 |          1 |      0.0767679 |      0.00110463 |     0.00219425 | {"pre
+-------------------+----+----+-----+------------+----------------+-----------------+----------------+------
5 rows in set (0.0005 sec)
```

- In the following example, a ONNX model for regression is imported. Then, the model is used to generate predictions, a score, and prediction explainers for a dataset in MySQL AI.

```
mysql> SET @model = 'sklearn_pipeline_regression_2_onnx';
Query OK, 0 rows affected (0.0003 sec)

mysql> SET @model_metadata = JSON_OBJECT('task','regression', 'onnx_outputs_info',JSON_OBJECT('predictions_n
Query OK, 0 rows affected (0.0003 sec)

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_sklearn_pipeline_regression_2, @model_metadata, @model);
Query OK, 0 rows affected (1.0652 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5141 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.regression_2_table', @model, 'mlcorpus.predictions', NULL);
Query OK, 0 rows affected (0.8902 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+-------------------+----+----+-----+--------+------------+------------------------------------------------
| _4aad19ca6e_pk_id | f1 | f2 | f3  | target | Prediction | ml_results
+-------------------+----+----+-----+--------+------------+------------------------------------------------
|                 1 | a  | 20 | 1.2 |   22.4 |     22.262 | {"predictions": {"prediction": 22.26203918457031
|                 2 | b  | 21 | 3.6 |   32.9 |    32.4861 | {"predictions": {"prediction": 32.48611450195312
|                 3 | c  | 19 | 7.8 |   56.8 |    56.2482 | {"predictions": {"prediction": 56.24815368652344
|                 4 | d  | 18 |   9 |   31.8 |       31.8 | {"predictions": {"prediction": 31.80000114440918
|                 5 | e  | 17 | 3.6 |   56.4 |    55.9861 | {"predictions": {"prediction": 55.98611450195312
+-------------------+----+----+-----+--------+------------+------------------------------------------------
5 rows in set (0.0005 sec)

mysql> CALL sys.ML_SCORE('mlcorpus.regression_2_table','target', @model, 'r2', @score, NULL);
Query OK, 0 rows affected (0.8688 sec)

mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.9993192553520203 |
+--------------------+
1 row in set (0.0003 sec)

mysql> CALL sys.ML_EXPLAIN('mlcorpus.regression_2_table', 'target', @model, JSON_OBJECT('model_explainer', '
       JSON_ARRAY('f1'), 'prediction_explainer', 'shap'));
```

```
Query OK, 0 rows affected (9.9860 sec)

m
mysql> CALL sys.ML_EXPLAIN_TABLE('mlcorpus.regression_2_predict', @model, 'mlcorpus.explanations', JSON_O
Query OK, 0 rows affected (8.2625 sec)

mysql> SELECT * FROM mlcorpus.explanations;
+-------------------+----+----+-----+------------+-----------------+----------------+----------------+---
| _4aad19ca6e_pk_id | f1 | f2 | f3  | Prediction | f1_attribution | f2_attribution | f3_attribution | ml_
+-------------------+----+----+-----+------------+-----------------+----------------+----------------+---
|                 1 | a  | 20 | 1.2 |     22.262 |        -10.7595 |       -4.25162 |       -2.48331 | {"
|                 2 | b  | 21 | 3.6 |    32.4861 |         2.33657 |       -8.50325 |        -1.1037 | {"
|                 3 | c  | 19 | 7.8 |    56.2482 |         14.8361 |              0 |        1.65554 | {"
|                 4 | d  | 18 |   9 |       31.8 |        -15.2433 |        4.25162 |        3.03516 | {"
|                 5 | e  | 17 | 3.6 |    55.9861 |         8.83008 |        8.50325 |        -1.1037 | {"
+-------------------+----+----+-----+------------+-----------------+----------------+----------------+---
5 rows in set (0.0006 sec)
```

- An example with task set to `NULL`.

```
mysql> SET @model = 'tensorflow_recsys_onnx';

mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode_tensorflow_recsys, NULL, @model);
Query OK, 0 rows affected (1.0037 sec)

mysql> CALL sys.ML_MODEL_LOAD(@model, NULL);
Query OK, 0 rows affected (0.5116 sec)

mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.recsys_predict', @model, 'mlcorpus.predictions', NULL);
Query OK, 0 rows affected (0.8271 sec)

mysql> SELECT * FROM mlcorpus.predictions;
+-------------------+---------+-------------+---------------------------+-------------------------------
| _4aad19ca6e_pk_id | user_id | movie_title | Prediction                | ml_results
+-------------------+---------+-------------+---------------------------+-------------------------------
|                 1 | a       | A           | {"output_1": ["0.7558"]} | {"predictions": {"prediction":
|                 2 | b       | B           | {"output_1": ["1.0443"]} | {"predictions": {"prediction":
|                 3 | c       | A           | {"output_1": ["0.8483"]} | {"predictions": {"prediction":
|                 4 | d       | B           | {"output_1": ["1.2986"]} | {"predictions": {"prediction":
|                 5 | e       | C           | {"output_1": ["1.1568"]} | {"predictions": {"prediction":
+-------------------+---------+-------------+---------------------------+-------------------------------
5 rows in set (0.0005 sec)
```

What's Next

- Review how to Create a Machine Learning Model.
- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.7.8 Analyzing Data Drift

MySQL AI includes data drift detection for classification and regression models.

Before You Begin

- Review how to Create a Machine Learning Model.
- Review use cases for Classification Data and Regression Analysis.

Data Drift Detection Overview

Machine learning typically makes an assumption that the training data and test data are similar. Over time, the similarity between the training data and the test data can decrease. This is known as data drift.

You can monitor data drift in the model catalog and when running the ML_PREDICT_ROW and ML_PREDICT_TABLE routines.

For the model catalog, the model_metadata column includes the training_drift_metric JSON object literal, which contains mean and variance numeric values. See Model Metadata.

mean and variance indicate the quality of the trained drift detector, and both values should be low. The more important value is mean, and if it is greater than 1.0, then drift evaluation for the test results might not be reliable.

For the ML_PREDICT_ROW and ML_PREDICT_TABLE routines, the *options* parameter includes the additional_details boolean value. If this option is enabled, the ml_results column includes the drift JSON object literal, which contains the metric numeric value and the attribution_percent JSON object literal.

- metric indicates the similarity between training and test data. A low value indicates similar values. A value grater than 1.0 indicates data drift, and the prediction results are questionable.
- attribution_percent indicates the top three features that contribute to data drift for each result. The higher the percentage value, the greater the contribution.

Workflow to Analyze Data Drift

The workflow to analyze data drift includes the following:

1. Run ML_TRAIN to train the machine learning model with either the classification or regression task.
2. When training is complete, query the model_metadata column and review the mean and variance values.
3. Run the ML_PREDICT_ROW or ML_PREDICT_TABLE routines on the trained model with the additional_details option set to true.
4. Review the drift parameter in ml_results.

Analyzing Data Drift in Model Metadata

To analyze data drift in model metadata:

1. Train the model with ML_TRAIN.

   ```
   mysql> CALL sys.ML_TRAIN('table_name', 'target_column_name', JSON_OBJECT('task', 'task_name'), @variable);
   ```

   Replace *table_name*, *target_column_name*, *task_name*, and *variable* with your own values. For example:

   ```
   mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cer
   ```

   Where:

   - census_data.census_train is the fully qualified name of the table that contains the training dataset (*schema_name.table_name*).
   - revenue is the name of the target column, which contains ground truth values.
   - JSON_OBJECT('task', 'classification') specifies the machine learning task type.

- `@census_model` is the name of the user-defined session variable that stores the model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. For example, `@my_model`. Learn more about Model Handles.

2. Query the `model_metadata` column from the model catalog. Optionally, use `JSON_PRETTY` to view the output in an easily readable format.

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=model_h
```

Replace *`user1`* with your own user name and *`model_handle`* with your own model handle. For example:

```
mysql> SELECT JSON_PRETTY(model_metadata) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census
+-----------------------------------------------------------+
| JSON_PRETTY(model_metadata)                               |
+-----------------------------------------------------------+
| {
  "task": "classification",
  "notes": null,
  "chunks": 1,
  "format": "HWMLv2.0",
  "n_rows": 100,
  "status": "Ready",
  "options": {
    "task": "classification",
    "model_explainer": "permutation_importance",
    "prediction_explainer": "permutation_importance"
  },
  "n_columns": 14,
  "column_names": [
    "age",
    "workclass",
    "fnlwgt",
    "education",
    "education-num",
    "marital-status",
    "occupation",
    "relationship",
    "race",
    "sex",
    "capital-gain",
    "capital-loss",
    "hours-per-week",
    "native-country"
  ],
  "contamination": null,
  "model_quality": "high",
  "training_time": 73.90254211425781,
  "algorithm_name": "RandomForestClassifier",
  "training_score": -0.35963335633277893,
  "build_timestamp": 1744377124,
  "n_selected_rows": 80,
  "training_params": {
    "recommend": "ratings",
    "force_use_X": false,
    "recommend_k": 3,
    "remove_seen": true,
    "ranking_topk": 10,
    "lsa_components": 100,
    "ranking_threshold": 1,
    "feedback_threshold": 1
  },
  "train_table_name": "census_data.census_train",
  "model_explanation": {
```

```
    "permutation_importance": {
      "age": -0.0057,
      "sex": 0.0002,
      "race": 0.0001,
      "fnlwgt": 0.0103,
      "education": 0.0108,
      "workclass": 0.0189,
      "occupation": 0.0,
      "capital-gain": 0.0304,
      "capital-loss": 0.0,
      "relationship": 0.0195,
      "education-num": 0.0152,
      "hours-per-week": 0.0235,
      "marital-status": 0.0099,
      "native-country": 0.0
    }
  },
  "n_selected_columns": 11,
  "target_column_name": "revenue",
  "optimization_metric": "neg_log_loss",
  "selected_column_names": [
    "age",
    "capital-gain",
    "education",
    "education-num",
    "fnlwgt",
    "hours-per-week",
    "marital-status",
    "race",
    "relationship",
    "sex",
    "workclass"
  ],
  "training_drift_metric": {
    "mean": 0.3535,
    "variance": 0.0597
  }
} |
+-----------------------------------------------------------+
1 row in set (0.0009 sec)
```

Where:

- `JSON_PRETTY` displays the information in an easily readable format.
- `ML_SCHEMA_user1.MODEL_CATALOG` refers to the model catalog name. Replace `user1` with your own user name.
- `model_handle` refers to the session variable for the trained model, `@census_model`. Learn more about Model Handles.

For `training_drift_metric`, the output generates a `mean` value of 0.3535 and a `variance` value of 0.0597, which indicates acceptable data drift.

Analyzing Data Drift Detection with ML_PREDICT_TABLE

To analyze data drift detection with a table of predictions:

1. If not done already, train the model to use. See Analyzing Data Drift in Model Metadata.
2. Load the trained model. Update `@census_model` with your own session variable for the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

3. Run `ML_PREDICT_TABLE` to generate a table of predictions.

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);
```

Replace *table_name*, *model_handle*, *output_table_name*),and *options* with your own values. For example:

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.`census_test`', @census_model, 'census_data.`census_test_
```

Where:

- `census_data.census_test` is the fully qualified name of the test dataset table (*database_name.table_name*).
- `@census_model` is the session variable that contains the model handle. See Work with Model Handles.
- `census_data.census_test_predictions` is the output table where predictions are stored.
- `JSON_OBJECT` includes the `additional_details` option set to `true`, so `ml_results` includes values for `metric` and `attribution_percent`.

4. Since a `metric` value over 1.0 indicates data drift, query rows in the output table that only have a metric value over 1.0.

```
mysql> SELECT ml_results FROM table_name WHERE JSON_EXTRACT(ml_results, '$.drift.metric') > 1.0;
```

Replace *table_name* with your own value. For example:

```
mysql> SELECT ml_results FROM census_test_predictions WHERE JSON_EXTRACT(ml_results, '$.drift.metric')
+--------------------------------------------------------------------------------------------------------
| ml_results
+--------------------------------------------------------------------------------------------------------
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.67, ">50K": 0.33}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.9, ">50K": 0.1}, "drift": {"metric
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.99, ">50K": 0.01}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.78, ">50K": 0.22}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.97, ">50K": 0.03}, "drift": {"metr
| {"predictions": {"revenue": ">50K"}, "probabilities": {"<=50K": 0.32, ">50K": 0.68}, "drift": {"metri
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.96, ">50K": 0.04}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.89, ">50K": 0.11}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.91, ">50K": 0.09}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.78, ">50K": 0.22}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.89, ">50K": 0.11}, "drift": {"metr
| {"predictions": {"revenue": "<=50K"}, "probabilities": {"<=50K": 0.62, ">50K": 0.38}, "drift": {"metr
+--------------------------------------------------------------------------------------------------------
12 rows in set (0.0014 sec)
```

The output displays the rows with high metric values (> 1.0), indicating data drift.

Analyzing Data Drift Detection with ML_PREDICT_ROW

To analyze data drift detection with one or more rows of predictions:

1. If not done already, train the model to use. See Analyzing Data Drift in Model Metadata.

2. Load the trained model. Update `@census_model` with your own session variable for the trained model.

```
mysql> CALL sys.ML_MODEL_LOAD(@census_model, NULL);
```

3. Run `ML_PREDICT_ROW` to generate predictions for a defined number of rows.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_name",
```

The following example generates predictions for three rows of the table. The output is similar to the previous example.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_test.`age`,
 "workclass", census_test.`workclass`,
 "fnlwgt", census_test.`fnlwgt`,
 "education", census_test.`education`,
 "education-num", census_test.`education-num`,
 "marital-status", census_test.`marital-status`,
 "occupation", census_test.`occupation`,
 "relationship", census_test.`relationship`,
 "race", census_test.`race`,
 "sex", census_test.`sex`,
 "capital-gain", census_test.`capital-gain`,
 "capital-loss", census_test.`capital-loss`,
 "hours-per-week", census_test.`hours-per-week`,
 "native-country", census_test.`native-country`),
 @census_model, JSON_OBJECT('additional_details', TRUE))FROM census_data.census_test LIMIT 3;
+---------------------------------------------------+
| sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_test.`age`,
"workclass", census_test.`workclass`,
"fnlwgt", census_test.`fnlwgt`,
"education", census_test.`education`,
"education-num", census_test.`education-num`,
"ma                                                |
+--------------------------------------------------+
|{                                                 |
|    "age": 37,                                    |
|    "sex": "Male",                                |
|    "race": "White",                              |
|    "fnlwgt": 99146,                              |
|    "education": "Bachelors",                     |
|    "workclass": "Private",                       |
|    "Prediction": "<=50K",                        |
|    "ml_results": {                               |
|        "drift": {                                |
|            "metric": 0,                          |
|            "attribution_percent": {              |
|                "age": 0,                         |
|                "fnlwgt": 46.67,                  |
|                "capital-gain": 0}},              |
|        "predictions": {                          |
|            "revenue": "<=50K"},                  |
|        "probabilities": {                        |
|            ">50K": 0.42,                         |
|            "<=50K": 0.58}},                      |
|    "occupation": "Exec-managerial",              |
|    "capital-gain": 0,                            |
|    "capital-loss": 1977,                         |
|    "relationship": "Husband",                    |
|    "education-num": 13,                          |
|    "hours-per-week": 50,                         |
|    "marital-status": "Married-civ-spouse",       |
|    "native-country": "United-States"}           |
|{                                                 |
|    "age": 34,                                    |
|    "sex": "Male",                                |
|    "race": "White",                              |
|    "fnlwgt": 27409,                              |
|    "education": "9th",                           |
|    "workclass": "Private",                       |
```

```
|     "Prediction": "<=50K",                           |
|     "ml_results": {                                  |
|         "drift": {                                   |
|             "metric": 0.1,                           |
|             "attribution_percent": {                 |
|                 "fnlwgt": 25,                        |
|                 "education": 33.31,                  |
|                 "workclass": 16.22}},                |
|         "predictions": {                             |
|             "revenue": "<=50K"},                     |
|         "probabilities": {                           |
|             ">50K": 0.24,                            |
|             "<=50K": 0.76}},                         |
|     "occupation": "Craft-repair",                    |
|     "capital-gain": 0,                               |
|     "capital-loss": 0,                               |
|     "relationship": "Husband",                       |
|     "education-num": 5,                              |
|     "hours-per-week": 50,                            |
|     "marital-status": "Married-civ-spouse",          |
|     "native-country": "United-States"}               |
|{                                                     |
|     "age": 30,                                       |
|     "sex": "Female",                                 |
|     "race": "White",                                 |
|     "fnlwgt": 299507,                                |
|     "education": "Assoc-acdm",                       |
|     "workclass": "Private",                          |
|     "Prediction": "<=50K",                           |
|     "ml_results": {                                  |
|         "drift": {                                   |
|             "metric": 0.26,                          |
|             "attribution_percent": {                 |
|                 "relationship": 21.36,               |
|                 "education-num": 28.33,              |
|                 "hours-per-week": 33.21}},           |
|         "predictions": {                             |
|             "revenue": "<=50K"},                     |
|         "probabilities": {                           |
|             ">50K": 0.01,                            |
|             "<=50K": 0.99}},                         |
|     "occupation": "Other-service",                   |
|     "capital-gain": 0,                               |
|     "capital-loss": 0,                               |
|     "relationship": "Unmarried",                     |
|     "education-num": 12,                             |
|     "hours-per-week": 40,                            |
|     "marital-status": "Separated",                   |
|     "native-country": "United-States"}               |
+------------------------------------------------------+
10 rows in set (6.8109 sec)
```

Where:

- The first `JSON_OBJECT` has output column names and key-value pairs of the columns in the trained table.
- `@census_model` is the session variable that contains the model handle. Learn more about Model Handles.
- The second `JSON_OBJECT` includes the `additional_details` option set to `true`, so `ml_results` includes values for `metric` and `attribution_percent`.

- census_data.census_test is the fully qualified name of the test dataset table (*database_name.table_name*).
- The LIMIT of 3 means that the output includes a maximum of three rows from the trained table.

The output allows you to review data drift values for the selected rows.

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.8 Monitoring the Status of AutoML

You can monitor the status of AutoML by querying the rapid_ml_status variable or by querying the ML_STATUS column of the performance_schema.rpd_nodes table.

Query the rapid_ml_status Variable

The rapid_ml_status variable provides the status of AutoML. Possible values are ON and OFF.

- ON: AutoML is up and running.
- OFF: AutoML is down.

The following example queries the rapid_ml_status status variable directly.

```
mysql> SHOW GLOBAL STATUS LIKE 'rapid_ml_status';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| rapid_ml_status | ON    |
+-----------------+-------+
```

The following example queries the rapid_ml_status status through the performance_schema.global_status table.

```
mysql> SELECT VARIABLE_NAME, VARIABLE_VALUE
       FROM performance_schema.global_status
       WHERE VARIABLE_NAME LIKE 'rapid_ml_status';
+-----------------+----------------+
| VARIABLE_NAME   | VARIABLE_VALUE |
+-----------------+----------------+
| rapid_ml_status | ON             |
+-----------------+----------------+
```

Query the ML_STATUS Column

The MySQL AI plugin writes AutoML status information to the ML_STATUS column of the performance_schema.rpd_nodes table after each AutoML query. Possible values include:

- UNAVAIL_MLSTATE: AutoML is not available.
- AVAIL_MLSTATE: AutoML is available.
- DOWN_MLSTATE: AutoML is down.

ML_STATUS is reported for each node.

You can query the ML_STATUS column of the performance_schema.rpd_nodes table.

To following example retrieves ID, STATUS, and ML_STATUS for each node from the performance_schema.rpd_nodes table:

```
mysql> SELECT ID, STATUS, ML_STATUS FROM performance_schema.rpd_nodes;
+----+---------------+---------------+
| ID | STATUS        | ML_STATUS     |
+----+---------------+---------------+
|  0 | AVAIL_RNSTATE | AVAIL_MLSTATE |
+----+---------------+---------------+
```

Resolve a Down Status for AutoML

If rapid_ml_status is OFF or ML_STATUS reports DOWN_MLSTATE for any node, you can restart the MySQL server and Cluster. Be aware that restarting interrupts any analytics queries that are running.

See the following to learn more:

- Managing MySQL Server with systemd
- A Quick Guide to Using the MySQL Yum Repository

What's Next

- Review Machine Learning Use Cases to create machine learning models with sample datasets.

4.9 AutoML Limitations

The following limitations apply to AutoML.

Text Handling Limitations

- AutoML only supports datasets in the English language.
- AutoML does not support text columns with NULL values.
- AutoML does not support a text target column.
- AutoML does not support recommendation tasks with a text column.
- For the forecasting task, endogenous_variables cannot be text.

Account Name Limitations

- The ML_TRAIN routine does not support MySQL user names that contain a period. For example, a user named 'joe.smith'@'%' cannot run the ML_TRAIN routine. The model catalog schema created by the ML_TRAIN procedure incorporates the user name in the schema name (for example., ML_SCHEMA_joesmith), and a period is not a permitted schema name character.

Memory Limitations

- The table used to train a model cannot exceed 10 GB, 100 million rows, or 1017 columns.

Routine and Query Limitations

- ML_EXPLAIN_TABLE and ML_PREDICT_TABLE are compute intensive processes, with ML_EXPLAIN_TABLE being the most compute intensive. Limiting operations to batches of 10 to 100 rows by splitting large tables into smaller tables is recommended. Use batch processing with the `batch_size` option. See the following to learn more:

  - ML_PREDICT_TABLE
  - ML_EXPLAIN_TABLE

- `ML_EXPLAIN`, `ML_EXPLAIN_ROW`, and `ML_EXPLAIN_TABLE` routines limit explanations to the 100 most relevant features.
- The `ML_PREDICT_TABLE` `ml_results` column contains the prediction results and the data. This combination must be less than 65,532 characters.
- Concurrent MySQL AI analytics and AutoML queries are not supported. An AutoML query must wait for MySQL AI analytics queries to finish, and vice versa. MySQL AI analytics queries are given priority over AutoML queries.

Chapter 5 AI-Powered Search and Content Generation

Table of Contents

This chapter describes the GenAI feature of MySQL AI.

5.1 About GenAI

The GenAI feature of MySQL AI lets you communicate with unstructured data using natural-language queries. It uses a familiar SQL interface which makes it is easy to use for content generation, summarization, and retrieval-augmented generation (RAG).

Using GenAI, you can perform natural-language searches in a single step using either in-database or external large language models (LLMs). All the elements that are necessary to use GenAI with proprietary data are integrated and optimized to work with each other.

Note

This chapter assumes that you are familiar with MySQL.

Key Features

- **In-Database LLM**

  GenAI uses a large language model (LLM) to enable natural language communication in multiple languages. You can use the capabilities of the LLM to search data as well as generate or summarize content. However, as this LLM is trained on public data, the responses to your queries are generated based on information available in the public data sources. To produce more relevant results, you can use the LLM capabilities with the vector store functionality to perform a vector search with RAG.

- **In-Database Vector Store**

  GenAI provides an inbuilt vector store that you can use to store enterprise-specific proprietary content available in your local filesystem, and perform vector-based similarity search across documents. Queries that you ask are automatically encoded with the same embedding model as the vector store without requiring any additional inputs or running a separate service. The vector store also provides valuable context for the LLM for RAG use cases.

- **Retrieval-Augmented Generation**

  GenAI retrieves content from the vector store and provides it as context to the LLM along with the query. This process of generating an augmented prompt is called retrieval-augmented generation (RAG), and it helps GenAI produce more contextually relevant, personalized, and accurate results.

- **GenAI Chat**

  This is an inbuilt chatbot that extends the LLMs capabilities as well as vector store and RAG functionalities of GenAI to let you ask multiple follow-up questions about a topic in a single session. You can use GenAI Chat to build customized chat applications by specifying custom settings, prompt, chat history length, and number of citations to be used for generating a response.

  GenAI Chat also provides a graphical interface integrated with the Visual Studio Code plugin for MySQL Shell.

- **Accelerated Vector-Based Query Processing**

  GenAI lets you run queries on tables that contain vector embeddings at an accelerated pace by offloading them to the MySQL AI Engine (AI engine). For more information, see About Accelerated Processing of Queries on Vector-Based Tables.

Benefits

GenAI lets you integrate generative AI into the applications, providing an integrated end-to-end pipeline including vector store generation, vector search with RAG, and an inbuilt chatbot.

Some key benefits of using the GenAI feature of MySQL AI are:

- The natural-language processing (NLP) capabilities of the LLMs let non-technical users have human-like conversations with the system in natural language.
- The in-database integration of LLM and embedding generation eliminates the need for using external solutions, and ensures the security of the proprietary content.
- The in-database integration of LLMs, vector store, and embedding generation simplifies complexity of applications that use these features.

What's Next

- Review the Supported Languages, Embedding Models, and LLMs.

5.2 Additional GenAI Requirements

To use the GenAI feature of MySQL AI, you must place the files that you want to ingest into the vector store in the local directory that you specified in the *Vector Store* tab in the MySQL AI installer. By default, this directory is set to `/var/lib/mysql-files`.

Vector store can ingest files in the following formats: PPTX, PPT, TXT, HTML, DOCX, DOC, and PDF. Each file can be up to 100 MB in size.

5.3 Required Privileges for using GenAI

To perform the following GenAI functions, ask the admin user to grant you the required privileges:

- To create a vector store, the `FILE` privilege is required:

```
mysql> GRANT FILE ON *.* TO 'user_name'@'%';
```

- To run the batch queries using `ML_GENERATE_TABLE`, `ML_RAG_TABLE`, and `ML_EMBED_TABLE`, the following privileges are required:

  - `SELECT` and `ALTER` privileges on the input table:

  ```
  mysql> GRANT SELECT, ALTER ON input_schema.input_table TO 'user_name'@'%';
  ```

  - `SELECT`, `INSERT`, `CREATE`, `DROP`, `ALTER`, `UPDATE` privileges on the schema where the output table is created.

  ```
  mysql> GRANT SELECT, INSERT, CREATE, DROP, ALTER, UPDATE ON output_schema.* TO 'user_name'@'%';
  ```

For more information, see Privileges Provided by MySQL and Default MySQL Privileges.

5.4 Supported LLM, Embedding Model, and Languages

This topic provides the list of languages that GenAI feature of MySQL AI supports and the embedding models as well as large language models (LLMs) that are available.

This topic contains the following sections:

- Viewing Available Models
- In-Database LLM
- In-Database Embedding Model
- Languages
- What's Next

Viewing Available Models

Once you are connected to the DB System, you can view the list of available models as shown below:

```
mysql> SELECT * FROM sys.ML_SUPPORTED_LLMS;
```

The output is similar to the following:

```
+----------+------------------------+-------------------+--------------------+---------------+
| provider | model_id               | availability_date | capabilities       | default_model |
+----------+------------------------+-------------------+--------------------+---------------+
| HeatWave | llama3.2-3b-instruct-v1 | 2025-05-20        | ["GENERATION"]     |             1 |
| HeatWave | all_minilm_l12_v2      | 2024-07-01        | ["TEXT_EMBEDDINGS"] |             0 |
| HeatWave | multilingual-e5-small  | 2024-07-24        | ["TEXT_EMBEDDINGS"] |             1 |
+----------+------------------------+-------------------+--------------------+---------------+
```

In-Database LLM

The following in-database LLM is available: `llama3.2-3b-instruct-v1`

In-Database Embedding Model

The following in-database embedding model is available:

- `all_minilm_l12_v2`
- `multilingual-e5-small`

Languages

GenAI feature of MySQL AI supports natural-language communication, ingesting documents, as well as generating text-based content in multiple languages. The quality of the generated text outputs depends on the training and ability of the LLM to work with the language.

Following is a list of languages supported by the GenAI:

- English (`en`)
- French (`fr`)
- German (`de`)
- Hindi (`hi`)
- Italian (`it`)
- Portuguese (`pt`)
- Spanish (`es`)
- Thai (`th`)

Note

To set the value of the `language` parameter in GenAI routines that support this parameter, do not use the language name to specify the language. Use the two-letter `ISO 639-1` code for the language instead. For example, to use French, use the `ISO 639-1` code for French, which is `fr`.

What's Next

- Learn how to perform the following tasks:
  - Generate Text-Based Content
  - Set Up a Vector Store
  - Generate Vector Embeddings
  - Perform a Vector Search
  - Start a Conversational Chat

5.5 Generating Text-Based Content

For generating text-based content and summarizing text, use the the `ML_GENERATE` routine uses the LLM to generate the text output.

The sections in this topic describe how to generate and summarize text-based content using the GenAI feature of MySQL AI.

5.5.1 Generating New Content

The following sections in this topic describe how to generate new text-based content using the GenAI feature of MySQL AI:

- Before You Begin
- Generating Content
- Running Batch Queries
- What's Next

Before You Begin

- Review the GenAI requirements and privileges.
- For Running Batch Queries, add the natural-language queries to a column in a new or existing table.

Generating Content

To generate text-based content using GenAI, perform the following steps:

1. To define your natural-language query, set the `@query` variable:

   ```
   mysql> SET @query="QueryInNaturalLanguage";
   ```

   Replace *`QueryInNaturalLanguage`* with a natural-language query of your choice. For example:

   ```
   mysql> SET @query="Write an article on Artificial intelligence in 200 words.";
   ```

2. To generate text-based content, pass the query to the LLM using the `ML_GENERATE` routine with the `task` parameter set to `generation`:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "generation", "model_id", "LLM", "language", "Language"));
   ```

   Replace the following:

   - *`LLM`*: LLM to use, which must be the same as the one you loaded in the previous step.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "generation", "model_id", "llama3.2-3b-instruct-v1", "language", "en"));
   ```

   Text-based content that is generated by the LLM in response to your query is printed as output. It looks similar to the text output shown below:

   ```
   | {"text": "\n**The Rise of Artificial Intelligence: Revolutionizing the Future**\n\nArtificial
   intelligence (AI) has been a topic of interest for decades, and its impact is becoming increasingly
   evident in various aspects of our lives. AI refers to the development of computer systems that can
   perform tasks that typically require human intelligence, such as learning, problem-solving, and
   decision-making.\n\nThe latest advancements in machine learning algorithms and natural language
   processing have enabled AI systems to become more sophisticated and efficient. Applications of AI
   are expanding rapidly across industries, including healthcare, finance, transportation, and education.
   For instance, AI-powered chatbots are being used to provide customer support, while self-driving
   ```

```
cars are being tested on roads worldwide.\n\nThe benefits of AI are numerous. It can automate
repetitive tasks, improve accuracy, and enhance productivity. Moreover, AI has the potential to solve
complex problems that were previously unsolvable by humans. However, there are also concerns about job
displacement and bias in AI decision-making.\n\nAs AI continues to evolve, it is essential to address
these challenges and ensure that its benefits are shared equitably among all stakeholders. With
continued investment in research and development, AI has the potential to transform industries and
improve lives worldwide. The future of work will be shaped by AI, and it's crucial to prepare for
this", "license": "Your use of this llama model is subject to your Oracle agreements and this llama
license agreement: https://downloads.mysql.com/docs/LLAMA_32_3B_INSTRUCT-license.pdf"} |
```

Running Batch Queries

To run multiple `generation` queries in parallel, use the `ML_GENERATE_TABLE` routine. This method is faster than running the `ML_GENERATE` routine multiple times.

To run the steps in this section, you can create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('Describe what is MySQL in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Artificial Intelligence in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Machine Learning in 50 words.');
```

To run batch queries using `ML_GENERATE_TABLE`, perform the following steps:

1. In the `ML_GENERATE_TABLE` routine, specify the table columns containing the input queries and for storing the generated text-based responses:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.O
   JSON_OBJECT("task", "generation", "model_id", "LLM", "language", "Language"));
   ```

   Replace the following:

   - *InputDBName*: the name of the database that contains the table column where your input queries are stored.
   - *InputTableName*: the name of the table that contains the column where your input queries are stored.
   - *InputColumn*: the name of the column that contains input queries.
   - *OutputDBName*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
   - *OutputTableName*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
   - *OutputColumn*: the name for the new column where you want to store the output generated for the input queries.
   - *LLM*: LLM to use, which must be the same as the LLM you loaded in the previous step.
   - *Language*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
   ```

```
JSON_OBJECT("task", "generation", "model_id", "llama3.2-3b-instruct-v1", "language", "en"));
```

2. View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {"text": "\nMySQL is an open-source relational database
management system (RDBMS) that allows users to store, manage,
and retrieve data in a structured format. It supports various
features like SQL queries, indexing, transactions, and security,
making it a popular choice for web applications, enterprise
software, and mobile apps development.",
"error": null,
"license": "Your use of this llama model is subject to your
Oracle agreements and this llama license agreement:
https://downloads.mysql.com/docs/LLAMA_32_3B_INSTRUCT-license.pdf"}
*************************** 2. row ***************************
    id: 2
Output: {"text": "\nArtificial Intelligence (AI) refers to the
development of computer systems that can perform tasks that
typically require human intelligence, such as learning,
problem-solving, and decision-making. AI uses algorithms and
data to mimic human thought processes, enabling machines to
analyze, reason, and interact with humans in increasingly
sophisticated ways.",
"error": null}
*************************** 3. row ***************************
    id: 3
Output: {"text": "\nMachine Learning (ML) is a subset of
Artificial Intelligence that enables systems to automatically
improve performance on a task without being explicitly programmed.
It involves training algorithms on data, allowing them to learn
patterns and make predictions or decisions based on new, unseen
data, without human intervention.",
"error": null}
```

The output table generated using the `ML_GENERATE_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the row in the output column.

If you created a new database for testing the steps in this section, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

To learn more about the available routine options, see ML_GENERATE_TABLE Syntax.

What's Next

Learn how to Summarize Existing Content.

5.5.2 Summarizing Content

The following sections in this topic describe how to summarize exiting content using the GenAI:

- Before You Begin
- Summarizing Content
- Running Batch Queries
- What's Next

Before You Begin

- Review the GenAI requirements and privileges.
- For Running Batch Queries, add the natural-language queries to a column in a new or existing table.

Summarizing Content

To summarize text, perform the following steps:

1. To define the text that you want to summarize, set the `@text` variable:

   ```
   mysql> SET @text="TextToSummarize";
   ```

   Replace *`TextToSummarize`* with the text that you want to summarize.

   For example:

   ```
   mysql> SET @text="Artificial Intelligence (AI) is a rapidly growing field that has the potential
   to revolutionize how we live and work. AI refers to the development of computer systems
   that can perform tasks that typically require human intelligence, such as visual perception,
   speech recognition, decision-making, and language translation. One of the most significant
   developments in AI in recent years has been the rise of machine learning, a subset of AI that
   allows computers to learn from data without being explicitly programmed. Machine learning
   algorithms can analyze vast amounts of data and identify patterns, making them increasingly
   accurate at predicting outcomes and making decisions. AI is already being used in a variety
   of industries, including healthcare, finance, and transportation. In healthcare, AI is being
   used to develop personalized treatment plans for patients based on their medical history and
   genetic makeup. In finance, AI is being used to detect fraud and make investment
   recommendations In transportation, AI is being used to develop self-driving cars and improve
   traffic flow. Despite the many benefits of AI, there are also concerns about its potential impact
   on society. Some worry that AI could lead to job displacement, as machines become more capable
   of performing tasks traditionally done by humans. Others worry that AI could be used for
   malicious ";
   ```

2. To generate the text summary, pass the original text to the LLM using the `ML_GENERATE` routine, with the `task` parameter set to `summarization`:

   ```
   mysql> SELECT sys.ML_GENERATE(@query,
   JSON_OBJECT("task", "summarization", "model_id", "LLM", "language", "Language"));
   ```

   Replace the following:

   - *`LLM`*: LLM to use, which must be the same as the one you loaded in the previous step. To view the lists of available LLMs, see In-Database LLM.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> SELECT sys.ML_GENERATE(@text,
   JSON_OBJECT("task", "summarization", "model_id", "llama3.2-3b-instruct-v1", "language", "en"));
   ```

   A text summary generated by the LLM in response to your query is printed as output. It looks similar to the text output shown below:

   ```
   | {"text": "\nHere is a concise summary of the text:\n\nArtificial Intelligence (AI) has the potential
   to revolutionize various aspects of life and work. AI systems can perform tasks that typically require
   human intelligence, such as visual perception, speech recognition, and decision-making. Machine learning,
   a subset of AI, enables computers to learn from data without explicit programming. AI is already being
   applied in healthcare, finance, and transportation, with applications including personalized treatment
   ```

```
plans, fraud detection, and self-driving cars. However, there are concerns about the impact of AI on
society, including job displacement and potential misuse for malicious purposes.", "license": "Your
use of this llama model is subject to your Oracle agreements and this llama license agreement:
https://downloads.mysql.com/docs/LLAMA_32_3B_INSTRUCT-license.pdf"} |
```

Running Batch Queries

To run multiple `summarization` queries in parallel, use the `ML_GENERATE_TABLE` routine. This method is faster than running the `ML_GENERATE` routine multiple times.

To run the steps in this section, create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES(
  CONCAT(
  'MySQL is a widely used open-source relational database management system or RDBMS that ',
  'is based on the SQL standard. It is designed to be highly scalable, reliable, and secure, ',
  'making it an ideal choice for businesses of all sizes. MySQL uses a client-server ',
  'architecture, where the server stores and manages the data, while clients connect to the ',
  'server to access and manipulate the data. The MySQL server can be installed on a variety ',
  'of operating systems, including Linux, Windows, and macOS. One of the key features of MySQL ',
  'is its support for stored procedures, which allow developers to create reusable blocks of ',
  'code that can be executed multiple times. This makes it easier to manage complex database ',
  'operations and reduces the amount of code that needs to be written. MySQL also supports ',
  'a wide range of data types, including integers, floating-point numbers, dates, and strings. ',
  'It also has built-in support for encryption, which helps to protect sensitive data from ',
  'unauthorized access. Another important feature of MySQL is its ability to handle large ',
  'amounts of data. It can scale horizontally by adding more servers to the cluster, or ',
  'vertically by upgrading the hardware.'
  )
);
mysql> INSERT INTO input_table (Input) VALUES(
  CONCAT(
  'Artificial Intelligence or AI refers to the simulation of human intelligence in machines ',
  'that are programmed to think and act like humans. The goal of AI is to create systems that ',
  'can function intelligently and independently, exhibiting traits associated with human ',
  'intelligence such as reasoning, problem-solving, perception, learning, and understanding ',
  'language. There are two main types of AI: narrow or weak AI, and general or strong AI. ',
  'Narrow AI is designed for a specific task and is limited in its abilities, while general ',
  'AI has the capability to understand or learn any intellectual task that a human being can. ',
  'AI technologies include machine learning, which allows systems to improve their performance ',
  'based on data, and deep learning, which involves the use of neural networks to model complex ',
  'patterns. Other AI techniques include natural language processing, robotics, and expert systems. ',
  'AI has numerous applications across various industries, including healthcare, finance, ',
  'transportation, and education. It has the potential to revolutionize the way we live and work ',
  'by automating tasks, improving efficiency, and enabling new innovations. However, there are ',
  'also concerns about the impact of AI on employment, privacy, and safety.'
  )
);
mysql> INSERT INTO input_table (Input) VALUES(
   CONCAT(
  'Machine learning is a subset of artificial intelligence that involves the development of ',
  'algorithms and statistical models that enable systems to improve their performance on a ',
  'specific task over time by learning from data. At its core, machine learning is about ',
  'using data to train machines to make predictions or decisions without being explicitly ',
  'programmed to do so. There are many different types of machine learning, including ',
  'supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, ',
  'the algorithm is trained on labeled data, meaning that the input data has been categorized or ',
  'classified by a human. The goal of supervised learning is to enable the machine to make predictions ',
  'based on this training data. Unsupervised learning, on the other hand, involves training the ',
  'algorithm on unlabeled data. In this case, the algorithm must identify patterns and relationships ',
  'in the data on its own. This type of learning is often used for tasks such as clustering or anomaly ',
  'detection. Reinforcement learning involves an agent interacting with an environment and learning by ',
  'trial and error. The agent receives feedback in the form of rewards or punishments, which it uses ',
```

```
  'to improve its behavior over time. This type of learning is often used in game playing or robotics.'
  )
);
```

To run batch queries using `ML_GENERATE_TABLE`, perform the following steps:

1. In the `ML_GENERATE_TABLE` routine, specify the table columns containing the input queries and for storing the generated text summaries:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.O
   JSON_OBJECT("task", "summarization", "model_id", "LLM", "language", "Language"));
   ```

   Replace the following:

   - *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
   - *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
   - *`InputColumn`*: the name of the column that contains input queries.
   - *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
   - *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
   - *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
   - *`LLM`*: LLM to use, which must be the same as the LLM you loaded in the previous step.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
   JSON_OBJECT("task", "summarization", "model_id", "llama3.2-3b-instruct-v1", "language", "en"));
   ```

2. View the contents of the output table:

   ```
   mysql> SELECT * FROM output_table\G
   *************************** 1. row ***************************
       id: 1
   Output: {"text": "\nHere is a concise summary:\n\nMySQL is an
   open-source relational database management system (RDBMS) that
   is widely used for its scalability, reliability, and security.
   It uses a client-server architecture and supports various
   operating systems. Key features include stored procedures for
   efficient code reuse, support for multiple data types,
   encryption for data protection, and the ability to handle large
   amounts of data through horizontal or vertical scaling.",
   "error": null,
   "license": "Your use of this llama model is subject to
   your Oracle agreements and this llama license agreement:
   https://downloads.mysql.com/docs/LLAMA_32_3B_INSTRUCT-license.pdf"}
   *************************** 2. row ***************************
       id: 2
   Output: {"text": "\nHere is a concise summary:\n\nArtificial
   ```

```
Intelligence (AI) refers to the simulation of human
intelligence in machines. There are two types: narrow AI
(limited to specific tasks) and general AI (capable of
understanding any intellectual task). AI technologies include
machine learning, deep learning, natural language processing,
robotics, and expert systems. With numerous applications
across industries, AI has the potential to revolutionize
various aspects of life, but also raises concerns about
employment, privacy, and safety.",
"error": null}
*************************** 3. row ***************************
    id: 3
Output: {"text": "\nHere is a concise summary:\n\nMachine
learning is a subset of AI that enables systems to improve
their performance over time by learning from data. It involves
developing algorithms and statistical models to make predictions
or decisions without explicit programming. There are three main
types: supervised, unsupervised, and reinforcement learning.
Supervised learning uses labeled data for prediction, while
unsupervised learning identifies patterns in unlabeled data.
Reinforcement learning involves an agent interacting with its
environment, receiving feedback to improve behavior through trial
and error.",
"error": null}
```

The output table generated using the `ML_GENERATE_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the row in the output column.

If you created a new database for testing the steps in this section, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

To learn more about the available routine options, see ML_GENERATE_TABLE Syntax.

What's Next

- Learn how to Set Up a Vector Store.
- Learn how to Generate Vector Embeddings.

5.6 Setting Up a Vector Store

Using the inbuilt vector store and retrieval-augmented generation (RAG), you can load and query unstructured documents stored in the local filesystem using natural language within the MySQL AI ecosystem.

The sections in this topic describe how to set up an inbuilt vector store.

5.6.1 About Vector Store and Vector Processing

This section describes the Vector Store functionality available with GenAI.

About Vector Store

A vector store is a relational database that lets you load unstructured data. It automatically parses unstructured data formats, which include PDF (including scanned PDF files), PPT, TXT, HTML, and DOC file formats, from the local filesystem. Then, it segments the parsed data, creates vector embeddings, and stores them for GenAI to perform semantic searches.

A vector store uses the native VECTOR data type to store unstructured data in a multidimensional space. Each point in a vector store represents the vector embedding of the corresponding data. Semantically similar data is placed closer in the vector space.

The large language models (LLMs) available in GenAI are trained on publicly available data. Therefore, the responses generated by these LLMs are based on publicly available information. To generate content relevant to your proprietary data, you must store your proprietary enterprise data, which has been converted to vector embeddings, in a vector store. This enables the in-database retrieval-augmented generation (RAG) system to perform a semantic search in the proprietary data stored in the vector stores to find appropriate content, which is then fed to the LLM to help it generate more accurate and relevant responses.

About Vector Processing

To create vector embeddings, GenAI uses in-database embedding models, which are encoders that converts sequence of words and sentences from documents into numerical representations. These numerical values are stored as vector embeddings in the vector store and capture the semantics of the data and relationships to other data.

A vector distance function measures the similarity between vectors by calculating the mathematical distance between two multidimensional vectors.

GenAI encodes your queries using the same embedding model that is used to encode the ingested data to create the vector store. It then uses the right distance function to find relevant content with similar semantic meaning from the vector store to perform RAG.

About Accelerated Processing of Queries on Vector-Based Tables

GenAI lets you run queries on tables that contain vector embeddings at an accelerated pace by offloading them to the MySQL AI Engine (AI engine). However, for query offload to be successful, the vector table must be offloaded to AI engine using the `SECONDARY_LOAD` clause with the `ALTER TABLE` statement, and the query (`SELECT` statement) must use at least one vector function in the `SELECT LIST`, `FILTER`, or `ORDER BY` expression. Additionally, only simple `SELECT` statements with `LIMIT_OFFSET`, `FILTER` and `ORDER BY` operations are offloaded to AI engine for accelerated processing.

To offload the vector table to AI engine, use the following statement:

```
mysql>ALTER TABLE tbl_name SECONDARY_LOAD;
```

Following are examples of queries that are offloaded to AI engine for accelerated processing:

- ```
  mysql>SELECT name, STRING_TO_VECTOR(embedding) FROM demo_table;
  ```
- ```
  mysql>SELECT name, STRING_TO_VECTOR(embedding) FROM demo_table limit 10;
  ```
- ```
  mysql>SELECT name, STRING_TO_VECTOR(embedding) FROM demo_table;
  ```
- ```
  mysql>SELECT name, ROUND(DISTANCE(@query_embedding_16, STRING_TO_VECTOR(embedding)), 4)
  AS distance FROM demo_table ORDER BY distance DESC;
  ```

Other SQL operations such as `JOIN`, `UNION`, `INTERSECT`, `GROUP BY`, `AGGREGATE`, `WINDOW`, and so on, are not supported for accelerated processing. Following are examples of queries that are not offloaded to AI engine for accelerated processing:

- Query containing no vector distance function:

```
mysql>SELECT COMPRESS(embedding) FROM demo_table1;
```

- Query containing `GROUP BY` or aggregates:

```
mysql>SELECT name, COUNT(DISTINCT embedding) FROM demo_table1 GROUP BY name;
```

- Query containing `JOIN` operation:

```
mysql>SELECT ROUND(DISTANCE(demo_table1.embedding, UNHEX("8679613f")), 4) from demo_table1 JOIN demo_tab
demo_table1.name = demo_table2.name;
```

About Optical Character Recognition

Optical Character Recognition (OCR) lets you extract and encode text from images stored in unstructured documents. The text extracted from images is converted into vector embeddings and stored in a vector store the same way regular text in unstructured documents is encoded and stored in a vector store.

OCR is enabled by default when you ingest files into a vector store.

However, when OCR is enabled, the loading process slows down because GenAI scans all images available in the files and pages of scanned documents that you are ingesting into the vector store. If OCR is not required for the documents that you are ingesting, you can disable OCR to speed up the loading process.

GenAI supports OCR in the following unstructured data formats: PDF (including scanned PDF files), DOC, DOCX, PPT, and PPTX. However, GenAI doesn't support OCR in TXT and HTML files. Images stored in TXT and HTML files are ignored while ingesting the files.

OCR in GenAI also has the following limitations:

- GenAI might not be able to extract and process the text from images with 100% accuracy. However, if there are minor character recognition errors, the overall meaning of the text is still preserved.
- In some cases, text-like figures in images might incorrectly be treated as regular text.
- GenAI doesn't support OCR for Scalable Vector Graphic (SVG) images in PDF files.

What's Next

Learn how to Ingest Files into a Vector Store.

5.6.2 Ingesting Files into a Vector Store

This section describes how to generate vector embeddings for files or folders stored in , and load the embeddings into a vector store table.

The following sections in this topic describe how to ingest files into a vector store:

- Before You Begin
- Ingesting Files into a Vector Store
- Cleaning Up
- What's Next

Before You Begin

- Review the GenAI requirements and privileges.
- Place the files that you want to load in the vector store directory that you specified in the MySQL AI installer.

  Vector store can ingest files in the following formats: PDF, PPTX, PPT, TXT, HTML, DOCX, and DOC.

  To test the steps in this topic, create a folder `demo-directory` inside the vector store director `/var/lib/mysql-files` for storing files that you want to ingest into the vector store. Then, download and place the MySQL HeatWave user guide PDF in the `demo-directory` folder.
- To create and store vector store tables using the steps described in this topic, you can create a new database `demo_db`:

  ```
  CREATE DATABASE demo_db;
  ```

Ingesting Files into a Vector Store

The `VECTOR_STORE_LOAD` routine creates and loads vector embeddings into the vector store. You can ingest the source files into the vector store using the following methods:

Perform the following steps:

1. To create the vector store table, use a new or existing database:

   ```
   mysql> USE DBName;
   ```

   Replace *`DBName`* with the database name.

   For example:

   ```
   mysql> USE demo_db;
   ```

2. Optionally, to specify a name for the vector store table and language to use, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "VectorStoreTableName", "language", "Language");
   ```

   Replace the following:

   - *`VectorStoreTableName`*: the name you want for the vector store table.
   - *`Language`*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT("table_name", "demo_embeddings", "language", "en");
   ```

   To learn more about the available routine options, see VECTOR_STORE_LOAD Syntax.

3. To import a file from the local filesystem and create a vector store table, use the `VECTOR_STORE_LOAD` routine:

   ```
   mysql> CALL sys.VECTOR_STORE_LOAD("file://FilePath", @options);
   ```

   Replace *`FilePath`* with the unique reference index (URI) of the files or directories to be ingested into the vector store. A URI is considered to be one of the following:

- A glob pattern, if it contains at least one unescaped `?` or `*` character.
- A prefix, if it is not a pattern and ends with a `/` character like a folder path.
- A file path, if it is neither a glob pattern nor a prefix.

Note

Ensure that the documents to be loaded are present in the directory that you specified for loading documents into the vector store during MySQL AI installation or using the `secure_file_priv` server system variable.

For example:

```
mysql> CALL sys.VECTOR_STORE_LOAD("file:///var/lib/mysql-files/demo-directory/heatwave-en.pdf", @option
```

This loads the specified file or files from the specified directory into the vector store table.

4. After the task is completed, verify that embeddings are loaded in the vector store table:

```
mysql> SELECT COUNT(*) FROM VectorStoreTableName;
```

For example:

```
mysql> SELECT COUNT(*) FROM demo_embeddings;
```

If you see a numerical value in the output, your embeddings are successfully loaded in the vector store table.

5. To view the details of the vector store table, use the following statement:

```
mysql> DESCRIBE demo_embeddings;
+-------------------+---------------+------+-----+---------+-------+
| Field             | Type          | Null | Key | Default | Extra |
+-------------------+---------------+------+-----+---------+-------+
| document_name     | varchar(1024) | NO   |     | NULL    |       |
| metadata          | json          | NO   |     | NULL    |       |
| document_id       | int unsigned  | NO   | PRI | NULL    |       |
| segment_number    | int unsigned  | NO   | PRI | NULL    |       |
| segment           | varchar(1024) | NO   |     | NULL    |       |
| segment_embedding | vector(384)   | NO   |     | NULL    |       |
+-------------------+---------------+------+-----+---------+-------+
```

Cleaning Up

If you created a new database for testing the steps in this topic, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

What's Next

- Learn how to Update the Vector Store.
- Learn how to Perform Vector Search With Retrieval-Augmented Generation.
- Learn how to Start a Conversational Chat.

5.6.3 Updating a Vector Store

To keep up with the changes and updates in the documents in your local directory, you must delete and recreate the vector store table. This ensures that the responses generated by GenAI are up-to-date.

The following sections in this topic describe how to update a vector store:

- Before You Begin
- Deleting and Recreating the Vector Store
- What's Next

Before You Begin

Complete the steps to set up a vector store.

Deleting and Recreating the Vector Store

To delete and recreate the vector store table and vector embeddings, perform the following steps:

1. Delete the vector store table:

   ```
   mysql> DROP TABLE VectorStoreTableName;
   ```

   Replace `VectorStoreTableName` with the vector store table name.

2. To create new embeddings for the updated documents, repeat the steps to set up a vector store.

What's Next

- Learn how to Generate Vector Embeddings.
- Learn how to Perform Vector Search With Retrieval-Augmented Generation.

5.7 Generating Vector Embeddings

This section describes how to generate vector embeddings using the `ML_EMBED_ROW` routine. Vector embeddings are a numerical representation of the text that capture the semantics of the data and relationships to other data. You can pass the text string in the routine manually or use data from tables in your database. To embed multiple rows of text stored in a table in a single run, you can even run a batch query.

Using this method, you can create vector embedding tables that you can use to perform similarity searches using the `DISTANCE()` function, without setting up a vector store.

> **Note**
>
> This method does not support embedding unstructured data. To learn how to create vector embeddings for unstructured data, see Section 5.6, “Setting Up a Vector Store”.

This topic contains the following sections:

- Before You Begin
- Generating a Vector Embedding for Specified Text
- Running Batch Queries
- What's Next

Before You Begin

- Review the GenAI requirements and privileges.
- For Running Batch Queries, add the text that you want to embed to a column in a new or existing table.

Generating a Vector Embedding for Specified Text

To generate a vector embedding, perform the following steps:

1. To define the text that you want to encode, set the `@text` variable:

   ```
   mysql> SET @text="TextToEncode";
   ```

   Replace *`TextToEncode`* with the text that you want to encode. For example:

   ```
   mysql> SET @text="MySQL AI lets you communicate with unstructured data using natural-language queries."
   ```

2. To generate a vector embedding for the specified text, pass the text to the embedding model using the `ML_EMBED_ROW` routine:

   ```
   mysql> SELECT sys.ML_EMBED_ROW(@text, JSON_OBJECT("model_id", "EmbeddingModel")) into @text_embedding;
   ```

   Replace *`EmbeddingModel`* with ID of the embedding model you want to use. To view the lists of available embedding models, see In-Database Embedding Model.

   For example:

   ```
   mysql> SELECT sys.ML_EMBED_ROW(@text, JSON_OBJECT("model_id", "all_minilm_l12_v2")) into @text_embeddin
   ```

   The routine returns a `VECTOR`, and this command stores it in the `@text_embedding` variable.

3. Print the vector embedding stored in the `@text_embedding` variable:

   ```
   mysql> SELECT @text_embedding;
   ```

   The output, which is a binary representation of the specified text, looks similar to the following:

   ```
   -----------------------------------------------------+
   | 0x6F57203BBF1592BD11FA93BD9FEC9E3C0A43CABDF1102EBD8B0B07BCF7D39ABCDCBEC7BC21ACACBC416B3FBD7A8E13
   3CA954B23D3F428DBD9A9E863DAE3085BC7E68313DA6E9BE3C3BA2F3BC3B2DC4BDFBCDD4BD0F2B593D00D95CBC2B40E53B
   8ED4AEBDD9B5D8BC695F703C3534463C7D7ADABB0EA7613CA4B40C3D40DD4A3D88E05E3DBDD8C43CF6B0863CE450ACBC3D
   34B63C978D99BC1EC638BD929CC7BD734E98BC7B9BAB3C2F3A47BD147E203D88EABD3DCF18483D42D820BD25C59BBC9E4D
   ADBC7DEA643D071F02BDA843AFBC865E323C775BBC3D87B8163D69DDF13DEAE5083DDA23353D2BDBCFBD0858ADBD9520E5
   3C1070343DE8237D3D6FA7083D1591653D90C8CE3DE4BE34BC6681B73D5D3CA5BCC2EBC8BD9102A3BCBE0A8EBD1C0189BB
   29CF0F3E2AA2ACBD075834BCC85AE33C224F9CBD261FDF3C7B34033CB8FCB4BCE247663DA3C2963B598089BBAFA5EABCC5
   59FBBD38E72BBDD8705D3BBAB3693DEDD26C3DB9CDDC3C2E51333D1A58E13CC67C6B3CA068D63C3DD35B3DBF72BCBCCBCC
   16BD8276513DE1B4913DDF7B05BDE9C836BB1BFD02BDE3AFA5BDBFAA68BD7780EB3CA39EB13C9D8CCCBD6260BCBC4A339A
   BCFE3A90BDD00B333D0622AABC2C5D47BDF406FF3D5F142FBC598B083DB2BA12BC3650D3BB07223A3D3E33F53CB3F032BC
   5CC6303C9CC1B63DC56AF33B424554BD3DC116BDAD93303C2E4A0D3D5FF4903D414E7C3CA315943DF69C35BD96C8473DF4
   62D2BA24CF2BBCA4E340BCAD53C6BCC8FF333DDC55643D447FF1B9742F35BD14B2423BEC5E0EBCC76E02BC230A2C3D663A
   6EBD27E1F0BCE2FF523BC5AB9ABD6921B13CE5EBA93D03A30D3E752FEC3C04151ABB14B3CEBD578BA93D31853DBC0D9685
   BD961AC2BC006CE0BDA835723CDE2AA1BC39728C3D484790BD980186BD4017C1BCB61F44BBC0FB8E3BC29AC93C6E36003E
   9A0F7F3D0D23213ACE228C3CEE0ED5BDD77491BD0E5834BD6680CEBD512A173D41BCB5BB4ABDA63B7F5C1B3D2C2C013EA5
   A4913D5CACFEBC611BC8BDCA3520BC1CB2D83CFFD3DEBB11998ABC4181713D5EAC003D01CFBB3C9333113C960849BD0F05
   99BD7A5BC13D2472403D9AF94ABD0B1C983C9429D53B654A413D079AECBD1F991C3D0B4BCB3C47AFCCBD1709743B291C57
   3DF35C13BD17C317BD519292BD85FBB23DAB319D3C1AEDA73B82C7BD3C8B5183BD7DE38DBC6A2AD1BB83D1F03A01718DBD
   236543BB6D22803CFF69133CB485188906BFC1BC75FAF03B24FA01BEFBE83B3D04F3353C4D67933D7ADECBBCAC79AF3B58
   AB8F3DE3C3B6BB050580BD92720C3DB0199BBC8A4790BC0D09B4BCAEC2503C1B2FAEBC91C598BA5070223D0CB8C13D2B6E
   D7BD5301553D326ECBBD6A8825BD75DE6E3C38380EBCFCE7F6BC9329FB3B1F7B3ABDF51B403D59EE873C33078CBD8CB7A5
   3B8D26A63DE2633CBDDBBFCEBB7778A63C566E84BD4D66973CF29CDFBC6271523D800EDABC57CD03BD81DB563D2B0BC4BC
   EB1238BC724B16BEACC15D3D8B8247BC24AAF63B29E7823C6300F13B4703193D8BD9D6BDBDD5313D68A73DBC36DBC5B981
   ```

```
0B36BDF940953A4B3EB2BCF9984E3C3EDD3DBD8709C83CCDE4ACBB4B8387BD48CA133D7187893C38FB9FBBF1F50CBDB650
06BDA3397B3DADB05CBD22961A3D405E16BBDF5E45BAEFC8A53D71FCD0BCAEE96F3D74DA0B3D724DE03C72A1653D53AF18
BCCD4A623D92033ABAF3E6AE3D68757C3D086475BDB6F9B03C1836CE3CA9D8FF3C8BFFC53B8A9A10BC96308EBD20FB7C3C
68610FBD5881310B1B52163D5ED0353C432D26BC31320FBC4E1ECCBCAA24A7BD480988BCE0CCB43D667CEFBB865600BD56
E9FA3960BA59BDE7C40F3DF01782BD0981E0394E1C5FBC8EA1443923ED633D9F00483D662A87BD2A568D3DC376503D996B
4BBD1F59D7BC92216E3D448BE2BC728DEFBC8F75013BF481753D9B71213C26541ABD2B93B43B54ED8EBCF0F7423D54C42D
3D5DAB58BC1D488CBC35CE69BDC6298CBD60F3E5BC5F7B003EB703003EF76FD1BCAF25A6BD8857F43C232B743CA96406BC
CA3536BD12BEC83D90FB0BBDB6D09EBDAE549BBD3C4CE83B8AD9733D5B890DBD57D1643B6F84E2BC73CC8DBD782B3D3D67F
CD7BCE1071CBDA1C0313DB99B993CFA29A3BD |
```


Running Batch Queries

To encode multiple rows of text strings stored in a table column, in parallel, use the `ML_EMBED_TABLE` routine. This method is faster than running the `ML_EMBED_ROW` routine multiple times.

To run the steps in this section, create a new database `demo_db` and table `input_table`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('Describe what is MySQL in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Artificial Intelligence in 50 words.');
mysql> INSERT INTO input_table (Input) VALUES('Describe Machine Learning in 50 words.');
```

To run batch queries using `ML_EMBED_TABLE`, perform the following steps:

1. Call the `ML_EMBED_TABLE` routine:

   ```
   mysql> CALL sys.ML_EMBED_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.Outp
   JSON_OBJECT("model_id", "EmbeddingModel"));
   ```

   Replace the following:

   - *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
   - *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
   - *`InputColumn`*: the name of the column that contains input queries.
   - *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
   - *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
   - *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.
   - *`EmbeddingModel`*: ID of the embedding model to use. To view the lists of available embedding models, see In-Database Embedding Model.

   For example:

   ```
   mysql> CALL sys.ML_EMBED_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output",
   JSON_OBJECT("model_id", "all_minilm_l12_v2"));
   ```

2. View the contents of the output table:

```
mysql> SELECT * FROM output_table;
+----+-------------------------------------------------------------------------------------------
| id | Output
+----+-------------------------------------------------------------------------------------------
|  1 | 0x0151873DB06340BD66E860BD26DC8C3BF5110ABEBF69F83B8791B63D955C003D1DF996BDA2B8883D772CB4BD882B44
|  2 | 0xE0C75E3DA70A91BDDF3996BB292AD7BA623AD1BDFB55763C85248F3DE0A85B3D5B8165BD257FCD3DB3F854BD72B556
|  3 | 0xB1CAE33C469EBFBD36C704BD926DBDBC4824B4BD8847723D0A93273DD9ED8F3CB9B384BDC300A53DF37B0ABD131E0A
+----+-------------------------------------------------------------------------------------------
```

The output table generated using the `ML_EMBED_TABLE` routine contains an additional column called `details` for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in this additional column.

```
mysql> DESCRIBE output_table;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| id      | int          | NO   | PRI | 0       |       |
| Output  | vector(2048) | YES  |     | NULL    |       |
| details | json         | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
```

To specify the embedding model used to generate the vector embeddings, the routine adds the following comment for the VECTOR column in the output table:

```
'GENAI_OPTIONS=EMBED_MODEL_ID=EmbeddingModelID'
```

For example:

```
mysql> SHOW CREATE TABLE output_table;
+--------------+-------------------------------------------------------------------------------------
| Table        | Create Table
+--------------+-------------------------------------------------------------------------------------
| output_table | CREATE TABLE `output_table` (
  `id` int NOT NULL DEFAULT '0',
  `Output` vector(2048) DEFAULT NULL COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=minilm',
  `details` json DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-------------------------------------------------------------------------------------
```

This lets you use tables generated using this routine for context retrieval while running retrieval-augmented generation (RAG) as well as GenAI Chat.

3. If you created a new database for testing the steps in this section, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

What's Next

- Learn how to Use Your Own Embeddings With Retrieval-Augmented Generation.
- Learn how to Start a Conversational Chat.

5.8 Performing Vector Search with Retrieval-Augmented Generation

When you enter a query, GenAI performs a vector-based similarity search to retrieve similar content from the vector store and embedding tables available in the DB system. It provides the retrieved content as context to the LLM. This helps the LLM to produce more relevant and accurate results for your query. This process is called as retrieval-augmented generation (RAG).

You can use both inbuilt vector store tables and tables containing your own vector embeddings for running RAG with vector search.

The topics in this section describe how to perform RAG with vector search.

5.8.1 Running Retrieval-Augmented Generation

The `ML_RAG` routine runs retrieval-augmented generation which aims to generate more accurate responses for your queries.

For context retrieval, the `ML_RAG` routine uses the name of the embedding model used to embed the input query to find relevant vector store tables that contain vector embeddings from the same embedding model.

This topic contains the following sections:

- Before You Begin
- Retrieving Context and Generating Relevant Content
- Retrieving Context Without Generating Content
- Running Batch Queries
- Cleaning Up
- What's Next

Before You Begin

- Complete the steps to set up a vector store.

  The examples in this topic use the vector store table `demo_embeddings` created in the section Ingesting Files into a Vector Store.

- For Running Batch Queries, add the natural-language queries to a column in a new or existing table.

Retrieving Context and Generating Relevant Content

To enter a natural-language query, retrieve the context, and generate results using RAG, perform the following steps:

1. Optionally, to speed up vector processing, load the vector store table in MySQL AI Engine (AI engine):

   ```
   mysql> ALTER TABLE VectorStoreTableName SECONDARY_LOAD;
   ```

   Replace *`VectorStoreTableName`* with the name of the vector store table.

   For example:

   ```
   mysql> ALTER TABLE demo_db.demo_embeddings SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. To specify the table for retrieving the vector embeddings to use as context, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("DBName.VectorStoreTableName"),
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

Replace the following:

- *DBName*: the name of the database that contains the vector store table.
- *VectorStoreTableName*: the name of the vector store table.
- *Language*: the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store", JSON_ARRAY("demo_db.demo_embeddings"),
  "model_options", JSON_OBJECT("language", "en")
);
```

To learn more about the available routine options, see ML_RAG Syntax.

3. To define your natural-language query, set the `@query` variable:

```
mysql> SET @query="AddYourQuery";
```

Replace *AddYourQuery* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

4. To retrieve the augmented prompt, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

5. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

Text-based content that is generated by the LLM in response to your query is printed as output. The output generated by RAG is comprised of two parts:

- The text section contains the text-based content generated by the LLM as a response for your query.
- The citations section shows the segments and documents it referred to as context.

The output looks similar to the following:

```
| {
  "text": "\nAutoML (Automated Machine Learning) is a machine learning technique that automates the pro
  "license": "Your use of this llama model is subject to your Oracle agreements and this llama license
  "citations": [
    {
      "segment": "\"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learning that
      "distance": 0.0732,
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
      "segment": "}, {   \"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learni
      "distance": 0.0738,
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the process
      "distance": 0.0743,
```

```
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0743
  }
} |
```

To continue running more queries in the same session, repeat steps 3 to 5.

Retrieving Context Without Generating Content

To enter a natural-language query and retrieve the context without generating a response for the query, perform the following steps:

1. Optionally, to speed up vector processing, load the vector store table in the AI engine:

   ```
   mysql> ALTER TABLE VectorStoreTableName SECONDARY_LOAD;
   ```

   Replace *`VectorStoreTableName`* with the name of the vector store table.

   For example:

   ```
   mysql> ALTER TABLE demo_db.demo_embeddings SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. To specify the table for retrieving the vector embeddings and to skip generation of content, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("DBName.VectorStoreTableName"), "skip_generate
   ```

   Replace the following:

   - *`DBName`*: the name of the database that contains the vector store table.
   - *`VectorStoreTableName`*: the name of the vector store table.

   For example:

   ```
   mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("demo_db.demo_embeddings"), "skip_generate", t
   ```

3. To define your natural-language query, set the `@query` variable:

   ```
   mysql> SET @query="AddYourQuery";
   ```

   Replace *`AddYourQuery`* with your natural-language query.

   For example:

   ```
   mysql> SET @query="What is AutoML?";
   ```

4. To retrieve the augmented prompt, use the `ML_RAG` routine:

   ```
   mysql> CALL sys.ML_RAG(@query,@output,@options);
   ```

5. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

Semantically similar text segments used as content for the query and the name of the documents they were found in are printed as output.

The output looks similar to the following:

```
| {
  "citations": [
    {
      "segment": "\"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learning that
      "distance": 0.0732,
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
      "segment": "}, {   \"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learni
      "distance": 0.0738,
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the process
      "distance": 0.0743,
      "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0743
  }
} |
```

To continue running more queries in the same session, repeat steps 3 to 5.

Running Batch Queries

To run multiple RAG queries in parallel, use the `ML_RAG_TABLE` routine. This method is faster than running the `ML_RAG` routine multiple times.

To run the steps in this section, create a new table `input_table` in `demo_db`:

```
mysql> USE demo_db;
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave Lakehouse?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave AutoML?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave GenAI?');
```

To run batch queries using `ML_RAG_TABLE`, perform the following steps:

1. To specify the table for retrieving the vector embeddings to use as context, set the `@options` variable:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store", JSON_ARRAY("DBName.VectorStoreTableName"),
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

   Replace the following:

   - *`DBName`*: the name of the database that contains the vector store table.
   - *`VectorStoreTableName`*: the name of the vector store table.

- *Language*: the two-letter ISO 639-1 code for the language you want to use. Default language is en, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store", JSON_ARRAY("demo_db.demo_embeddings"),
  "model_options", JSON_OBJECT("language", "en")
);
```

To learn more about the available routine options, see ML_RAG_TABLE Syntax.

2. In the ML_RAG_TABLE routine, specify the table columns containing the input queries and for storing the generated outputs:

```
mysql> CALL sys.ML_RAG_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.Output
```

Replace the following:

- *InputDBName*: the name of the database that contains the table column where your input queries are stored.
- *InputTableName*: the name of the table that contains the column where your input queries are stored.
- *InputColumn*: the name of the column that contains input queries.
- *OutputDBName*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
- *OutputTableName*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *OutputColumn*: the name for the new column where you want to store the output generated for the input queries.

For example:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", @options);
```

3. View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {"text": "\nHeatWave Lakehouse is a feature of the HeatWave platform that enables query processing
"error": null,
"license": "Your use of this llama model is subject to your Oracle agreements and this llama license agreem
"citations": [
    {
        "segment": "----------------------+ |  1 | {\"text\": \" HeatWave Lakehouse is a feature of the He
        "distance": 0.0828,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
        "segment": "-----------------------------------------+ |  1 | {\"text\": \" HeatWave Lakehouse is
        "distance": 0.0863,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
```

```
        "segment": "The Lakehouse feature of HeatWave enables query processing on data in Object Storag
        "distance": 0.1028,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.1028}}
*************************** 2. row ***************************
    id: 2
Output: {"text": "\nHeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine le
"error": null,
"citations": [
    {
        "segment": "|  HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine
        "distance": 0.0561,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
        "segment": "HeatWave shapes and scaling, and all HeatWave AutoML makes it easy to use machine l
        "distance": 0.0573,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
        "segment": "HeatWave AutoML makes it easy to use machine learning, whether you are a novice use
        "distance": 0.0598,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.0598}}
*************************** 3. row ***************************
    id: 3
Output: {"text": "\nHeatWave GenAI is a feature of HeatWave that enables natural language communication
"error": null,
"citations": [
    {
        "segment": "4.1 HeatWave GenAI Overview HeatWave GenAI is a feature of HeatWave that lets you c
        "distance": 0.0521,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
        "segment": "Chapter 3, HeatWave AutoML. 1.4 HeatWave GenAI The HeatWave GenAI feature of HeatWa
        "distance": 0.0735,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    },
    {
        "segment": "HeatWave Chat also provides a graphical interface integrated with the Visual Studio
        "distance": 0.0781,
        "document_name": "/var/lib/mysql-files/demo-directory/heatwave-en.pdf"
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.0781}}
```

The output table generated using the `ML_RAG_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in the output column.

Cleaning Up

If you created a new database for testing the steps in this topic, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

What's Next

- Learn how to Use Your Own Embeddings With Retrieval-Augmented Generation.
- Learn how to Start a Conversational Chat.

5.8.2 Using Your Own Embeddings with Retrieval-Augmented Generation

GenAI lets you use tables containing your own vector embedding to run retrieval-augmented generation (RAG) with vector search. The `ML_RAG` and `ML_RAG_TABLE` routines let you specify the table column names to use as filters for finding relevant tables for context retrieval.

In addition to the specified column names, the `ML_RAG` and `ML_RAG_TABLE` routines use the name of the embedding model used to embed the input query to find relevant embedding tables for context retrieval.

Following sections in this topic describe how you can use your own embedding table for context retrieval:

- Before You Begin
- Using Embeddings From an Embedding Model Available in GenAI
- Using Embeddings From an Embedding Model Not Available in GenAI
- Using Your Embedding Table With a Vector Store Table
- Running Batch Queries
- Cleaning Up
- What's Next

Before You Begin

- Review the GenAI requirements and privileges.
- You can use a table that satisfies the following criteria:
  - To qualify as a valid embedding table, the table must contain the following columns:
    - A string column containing the text segments.
    - A vector column containing the vector embeddings of the text segments.
  - A comment on the vector column must specify the name of the embedding model used to generate the vector embeddings.

  Following is an example of a valid embedding table that can be used for context retrieval:

```
mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT,
demo_text TEXT,
string_embedding TEXT,
demo_embedding VECTOR (3) COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=demo_embedding_model',
primary key (id));
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.',
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('AI refers to the development of machines that can think and act like humans.', '[0,0,1]');
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on tasks by
mysql> UPDATE demo_table SET demo_embedding=STRING_TO_VECTOR(string_embedding);
mysql> ALTER TABLE demo_table DROP COLUMN string_embedding;
```

To learn how to generate vector embeddings and embedding tables using GenAI, see Generating Vector Embeddings.

- If you want to use an inbuilt vector store table along with your own embedding table, complete the steps to set up the vector store.
- For Running Batch Queries, add the natural-language queries to a column in a new or existing table. To use the name of an embedding model that is not available in GenAI for running RAG, also add the vector embeddings of the input queries to a column of the input table.
- To create and store the sample embedding tables required for running the steps in this topic, you can create and use a new database `demo_db`:

```
mysql> CREATE DATABASE demo_db;
mysql> USE demo_db;
```

Using Embeddings From an Embedding Model Available in GenAI

To use an embedding table containing vector embeddings from an embedding model that is available in GenAI, you can set the `vector_store_columns` parameter to specify the columns and column names used by the `ML_RAG` routine to filter tables for context retrieval. However, since the inbuilt vector store tables only use the predefined column names, if you change a column name used for filtering tables, the inbuilt vector store tables are filtered out and not used for context retrieval.

The example in this section uses the following table:

```
mysql> CREATE TABLE demo_minilm_table (id INT AUTO_INCREMENT, demo_text_column TEXT, primary key (id));
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.'
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('AI refers to the development of machines that can think and act like humans.');
mysql> INSERT INTO demo_minilm_table (demo_text_column)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on tasks by
mysql> CALL sys.ML_EMBED_TABLE('demo_db.demo_minilm_table.demo_text_column', 'demo_db.demo_minilm_table.dem
JSON_OBJECT('model_id', 'all_minilm_l12_v2'));
```

To run RAG, perform the following steps:

1. Optionally, to speed up vector processing, load the embedding table in the MySQL AI Engine (AI engine):

   ```
   mysql> ALTER TABLE EmbeddingTableName SECONDARY_LOAD;
   ```

   Replace *`EmbeddingTableName`* with the embedding table name.

   For example:

   ```
   mysql> ALTER TABLE demo_minilm_table SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. To change the column names to use to filter tables for context retrieval, then set the routine options as shown below:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorE
     "embed_model_id", "EmbeddingModelName",
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

Replace the following:

- *TextSegmentColumnName*: the name of the embedding table column that contains the text segments in natural language. Default value is `segment`.
- *VectorEmbeddingColumnName*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. Default value is `segment_embedding`.
- *EmbeddingModelName*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find relevant tables for context retrieval. Default value is `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

  For possible values, to view the list of available embedding models, see In-Database Embedding Model.
- *Language*: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "demo_text_column", "segment_embedding", "demo_embedding_c
  "embed_model_id", "all_minilm_l12_v2", "model_options", JSON_OBJECT("language", "en")
);
```

In this example, all embedding tables containing a string column `demo_text_column` and a vector column `demo_embedding_column`, which contains vector embeddings from `all_minilm_l12_v2`, are used for context retrieval.

Similarly, you can use the `vector_store_columns` parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- `document_name`: name of a column containing the document names. This column can be of any data type supported by MySQL. Default value is `document_name`.
- `document_id`: name of an integer column containing the document IDs. Default value is `document_id`.
- `metadata`: name of a JSON column containing additional table metadata. Default value is `metadata`.
- `segment_number`: name of an integer column containing the segment numbers. Default value is `segment_number`.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

3. To define your natural-language query, set the `@query` variable:

```
SET @query="AddYourQuery";
```

Replace *AddYourQuery* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

4. To retrieve the augmented prompt and generate the output, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

5. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

The output is similar to the following:

```
| {
  "text": "\nBased on the context, AutoML stands for Automated Machine Learning. It is a subset of AI t
  "license": "Your use of this llama model is subject to your Oracle agreements and this llama license
  "citations": [
    {
      "segment": "AI refers to the development of machines that can think and act like humans.",
      "distance": 0.733,
      "document_name": ""
    },
    {
      "segment": "ML is a subset of AI that uses algorithms and statistical models to improve performan
      "distance": 0.7375,
      "document_name": ""
    },
    {
      "segment": "MySQL is an open-source RDBMS that is widely used for its scalability, reliability, a
      "distance": 0.8234,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_minilm_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.8234
  }
} |
```

The `vector_store` section lists the name of the embedding table that is used to retrieve context for generating the output.

Using Embeddings From an Embedding Model Not Available in GenAI

To use a table containing vector embeddings from an embedding model that is not available in GenAI, the `ML_RAG` routine lets you provide the vector embedding of the input query and the name of the embedding model that you used to embed the input query as well as the vector embeddings stored in your embedding table. When you provide the vector embedding of the input query, the routine skips embedding the query and proceeds with the similarity search, context retrieval, and RAG. However, in this case, you cannot use the inbuilt vector store tables for context retrieval.

The example in this section uses the following table:

```
mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT,
demo_text TEXT,
string_embedding TEXT,
demo_embedding VECTOR (3) COMMENT 'GENAI_OPTIONS=EMBED_MODEL_ID=demo_embedding_model',
primary key (id));
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.', '[
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('AI refers to the development of machines that can think and act like humans.', '[0,0,1]');
mysql> INSERT INTO demo_table (demo_text, string_embedding)
VALUES('ML is a subset of AI that uses algorithms and statistical models to improve performance on tasks by le
mysql> UPDATE demo_table SET demo_embedding=STRING_TO_VECTOR(string_embedding);
mysql> ALTER TABLE demo_table DROP COLUMN string_embedding;
```

To run RAG using a table that contains vector embeddings from an embedding model that is not available in GenAI, perform the following steps:

1. Optionally, to speed up vector processing, load the embedding table in the AI engine:

   ```
   mysql> ALTER TABLE EmbeddingTableName SECONDARY_LOAD;
   ```

   Replace *EmbeddingTableName* with the embedding table name.

   For example:

   ```
   mysql> ALTER TABLE demo_table SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. Provide the vector embedding of the input query:

   ```
   SET @query_embedding = to_base64(string_to_vector('VectorEmbeddingOfTheQuery'));
   ```

   Replace *VectorEmbeddingOfTheQuery* with the vector embedding of your input query.

   For example:

   ```
   mysql> SET @query_embedding = to_base64(string_to_vector('[0,1,0]'));
   ```

3. To specify column names for the `ML_RAG` routine to find relevant tables for context retrieval, set the routine options:

   ```
   mysql> SET @options = JSON_OBJECT(
     "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorEmbed
     "embed_model_id", "EmbeddingModelName",
     "query_embedding", @query_embedding,
     "model_options", JSON_OBJECT("language", "Language")
   );
   ```

   Replace the following:

   - *TextSegmentColumnName*: the name of the embedding table column that contains the text segments in natural language.
   - *VectorEmbeddingColumnName*: the name of the embedding table column that contains vector embeddings of the natural-language text segments.
   - *EmbeddingModelName*: the name of the embedding model that you used to generate the vector embeddings for the input query and embedding tables.

- *Language*: the two-letter ISO 639-1 code for the language you want to use for generating the output. The model_option option parameter language is required only if you want to use a language other than English. Default language is en, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "demo_text", "segment_embedding", "demo_embedding"),
  "embed_model_id", "demo_embedding_model",
  "query_embedding", @query_embedding,
  "model_options", JSON_OBJECT("language", "en")
);
```

In this example, embedding tables containing a string column demo_text and a vector column demo_embeddings which contains vector embeddings from demo_embedding_model are used for context retrieval.

Similarly, you can use the vector_store_columns parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- document_name: name of a column containing the document names. This column can be of any data type supported by MySQL.
- document_id: name of an integer column containing the document IDs.
- metadata: name of a JSON column containing additional table metadata.
- segment_number: name of an integer column containing the segment numbers.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

4. To define your natural-language query, set the @query variable:

```
SET @query="AddYourQuery";
```

Replace *AddYourQuery* with your natural-language query.

For example:

```
mysql> SET @query="What is AutoML?";
```

5. To retrieve the augmented prompt, use the ML_RAG routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

6. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

The output is similar to the following:

```
| {
  "text": "\nBased on the context, AutoML stands for Automated Machine Learning. It is a subset of AI t
  "license": "Your use of this llama model is subject to your Oracle agreements and this llama license
  "citations": [
    {
      "segment": "MySQL is an open-source RDBMS that is widely used for its scalability, reliability, a
      "distance": 0.0,
```

```
      "document_name": ""
    },
    {
      "segment": "ML is a subset of AI that uses algorithms and statistical models to improve performance o
      "distance": 0.2929,
      "document_name": ""
    },
    {
      "segment": "AI refers to the development of machines that can think and act like humans.",
      "distance": 1.0,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 1.0
  }
} |
```

The `vector_store` section lists the name of the embedding table that is used to retrieve context for generating the output.

Using Your Embedding Table With a Vector Store Table

By default, the `ML_RAG` routine uses all predefined columns and column names available in the inbuilt vector store table to filter tables for context retrieval. This means that if your embedding table does not contain all columns that are available in an inbuilt vector store table, then your embedding table is filtered out and is not used for context retrieval by the routine.

Therefore, if you want to use an inbuilt vector store table along with your own embedding table for context retrieval, your embedding table must satisfy the following additional requirements:

- Since the inbuilt vector store tables, use predefined column names, the column names in your embedding tables must match the predefined inbuilt vector store table column names as given below:
  - `segment`: name of the mandatory string column containing the text segments.
  - `segment_embedding`: name of the mandatory vector column containing the vector embeddings of the text segments.
  - `document_name`: name of the optional column containing the document names. This column can be of any data type supported by MySQL.
  - `document_id`: name of the optional integer column containing the document IDs.
  - `metadata`: name of the optional JSON column containing metadata for the table.
  - `segment_number`: name of the optional integer column containing segment number.
- The vector embeddings in your embedding table must be from the same embedding model as the vector store table.

The example in this section uses the vector store table `demo_embeddings` created in the section Ingesting Files into a Vector Store, which has been loaded into the AI engine, with the following table:

```
mysql> CREATE TABLE demo_e5_table (id INT AUTO_INCREMENT, segment TEXT, primary key (id));
mysql> INSERT INTO demo_e5_table (segment)
VALUES('MySQL is an open-source RDBMS that is widely used for its scalability, reliability, and security.');
mysql> INSERT INTO demo_e5_table (segment)
```

```
VALUES('AI refers to the development of machines that can think and act like humans.');
mysql> INSERT INTO demo_e5_table (segment)
VALUES('Machine learning is a subset of AI that uses algorithms and statistical models to improve performan
mysql> CALL sys.ML_EMBED_TABLE('demo_db.demo_e5_table.segment', 'demo_db.demo_e5_table.segment_embedding',
JSON_OBJECT('model_id', 'multilingual-e5-small'));
```

To run RAG using an inbuilt vector store table and your embedding table, perform the following steps:

1. Optionally, to speed up vector processing, load the embedding table in the AI engine:

   ```
   mysql> ALTER TABLE EmbeddingTableName SECONDARY_LOAD;
   ```

   Replace `EmbeddingTableName` with the embedding table name.

   For example:

   ```
   mysql> ALTER TABLE demo_e5_table SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. Set the routine options:

   - If your embedding table contains all the mandatory and optional columns as the inbuilt vector store table, then set the routine options as shown below:

     ```
     mysql> SET @options = JSON_OBJECT(
       "embed_model_id", "EmbeddingModelName",
       "model_options", JSON_OBJECT("language", "Language"
       )
     );
     ```

     - `EmbeddingModelName`: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find relevant tables for context retrieval. Default value is `multilingual-e5-small`.

       For possible values, to view the list of available embedding models, see In-Database Embedding Model.

     - `Language`: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

     For example:

     ```
     mysql> SET @options = JSON_OBJECT("embed_model_id", "multilingual-e5-small", "model_options", JSON_OB
     ```

   - If your embedding table contains the same mandatory columns as that of an inbuilt vector store table, similar to `demo_e5_table`, which are:

     - A text column with the name `segment`.

     - A vector column `segment_embedding`.

     Then, set the routine options as shown below:

     ```
     mysql> SET @options = JSON_OBJECT(
       "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embedding")
       "embed_model_id", "EmbeddingModelName",
       "model_options", JSON_OBJECT("language", "Language"
     ```

```
));
```

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embedding"),
  "embed_model_id", "multilingual-e5-small",
  "model_options", JSON_OBJECT("language", "en")
);
```

In this example, both embedding tables and vector store tables that contain a string column `segment` and a vector column `segment_embedding` which contains vector embeddings from `multilingual-e5-small` are used for context retrieval.

3. To define your natural-language query, set the `@query` variable:

```
SET @query="AddYourQuery";
```

   Replace *`AddYourQuery`* with your natural-language query.

   For example:

```
mysql> SET @query="What is AutoML?";
```

4. To retrieve the augmented prompt and generate the output, use the `ML_RAG` routine:

```
mysql> CALL sys.ML_RAG(@query,@output,@options);
```

5. Print the output:

```
mysql> SELECT JSON_PRETTY(@output);
```

   The output is similar to the following:

```
| {
  "text": "\nAutoML (Automated Machine Learning) is a machine learning technique that automates the process
  "license": "Your use of this llama model is subject to your Oracle agreements and this llama license agre
  "citations": [
    {
      "segment": "\"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learning that foc
      "distance": 0.0732,
      "document_name": ""
    },
    {
      "segment": "}, {   \"segment\": \"| {   \\\"text\\\": \\\" AutoML is a subfield of machine learning t
      "distance": 0.0738,
      "document_name": ""
    },
    {
      "segment": "| {   \"text\": \" AutoML is a machine learning technique that automates the process of s
      "distance": 0.0743,
      "document_name": ""
    }
  ],
  "vector_store": [
    "`demo_db`.`demo_embeddings`",
    "`demo_db`.`demo_e5_table`"
  ],
  "retrieval_info": {
    "method": "n_citations",
    "threshold": 0.0743
  }
```

```
} |
```

The `vector_store` section lists the names of the vector store table, `demo_embeddings`, and embedding table, `demo_e5_table` that are used to retrieve context for generating the output.

Running Batch Queries

To run multiple RAG queries in parallel, use the `ML_RAG_TABLE` routine. This method is faster than running the `ML_RAG` routine multiple times.

To run the steps in this section, you can use the same sample table `demo_e5_table` as section Using Your Embedding Table With a Vector Store Table, and create the following table to store input queries for batch processing:

```
mysql> CREATE TABLE input_table (id INT AUTO_INCREMENT, Input TEXT, primary key (id));
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave Lakehouse?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave AutoML?');
mysql> INSERT INTO input_table (Input) VALUES('What is HeatWave GenAI?');
```

To run batch queries using `ML_RAG_TABLE`, perform the following steps:

1. To specify column names for the `ML_RAG_TABLE` routine to find relevant tables for context retrieval, set the routine options:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "VectorE
  "embed_model_id", "EmbeddingModelName",
  "model_options", JSON_OBJECT("language", "Language")
);
```

Replace the following:

- *`TextSegmentColumnName`*: the name of the embedding table column that contains the text segments in natural language. If multiple tables contain a string column with the same name, they are all used for context retrieval. Default value is `segment`.

- *`VectorEmbeddingColumnName`*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. If multiple tables contain a vector column with the same name which contain embeddings from the specified embedding model, they are all used for context retrieval. Default value is `segment_embedding`.

- *`EmbeddingModelName`*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find tables generated using the same model for context retrieval. Default value is `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

- *`Language`*: the two-letter `ISO 639-1` code for the language you want to use for generating the output. The `model_option` option parameter `language` is required only if you want to use a language other than English. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example:

```
mysql> SET @options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "segment", "segment_embedding", "segment_embedding"),
  "embed_model_id", "multilingual-e5-small",
  "model_options", JSON_OBJECT("language", "en")
);
```

In this example, only embedding tables containing a string column demo_text and a vector column demo_embeddings which contains vector embeddings from multilingual-e5-small are used for context retrieval. Since the inbuilt vector store tables use predefined column names, if you change the column names to any value other than the default value, then the vector store tables are filtered out and are not used for context retrieval.

To learn more about the available routine options, see ML_RAG_TABLE Syntax.

Similarly, you can use the vector_store_columns parameter to specify the following column names for the routine to filter relevant tables for context retrieval:

- document_name: name of a column containing the document names. This column can be of data type supported by MySQL. Default value is document_name.
- document_id: name of an integer column containing the document IDs. Default value is document_id.
- metadata: name of a JSON column containing additional table metadata. Default value is metadata.
- segment_number: name of an integer column containing the segment numbers. Default value is segment_number.

Since these are optional columns, if these column values are not set, then the routine does not use these columns to filter tables.

If you are using an embedding model that is not available in GenAI, then you must also provide the vector embeddings of the input queries. You can specify name of the input table column that contains the vector embeddings of the input queries using the embed_column parameter. However, in this case, you cannot use the inbuilt vector store tables for context retrieval.

2. In the `ML_RAG_TABLE` routine, specify the table columns containing the input queries and for storing the generated outputs:

```
mysql> CALL sys.ML_RAG_TABLE("InputDBName.InputTableName.InputColumn", "OutputDBName.OutputTableName.Ou
```

Replace the following:

- *`InputDBName`*: the name of the database that contains the table column where your input queries are stored.
- *`InputTableName`*: the name of the table that contains the column where your input queries are stored.
- *`InputColumn`*: the name of the column that contains input queries.
- *`OutputDBName`*: the name of the database that contains the table where you want to store the generated outputs. This can be the same as the input database.
- *`OutputTableName`*: the name of the table where you want to create a new column to store the generated outputs. This can be the same as the input table. If the specified table doesn't exist, a new table is created.
- *`OutputColumn`*: the name for the new column where you want to store the output generated for the input queries.

For example:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", @options);
```

View the contents of the output table:

```
mysql> SELECT * FROM output_table\G
*************************** 1. row ***************************
    id: 1
Output: {"text": "\nHeatWave Lakehouse is a feature of the HeatWave platform that enables query process
"error": null,
"license": "Your use of this llama model is subject to your Oracle agreements and this llama license ag
"citations": [
    {
        "segment": "-----------------------+ |  1 | {\"text\": \" HeatWave Lakehouse is a feature of th
        "distance": 0.0828,
        "document_name": ""
    },
    {
        "segment": "------------------------------------------+ |  1 | {\"text\": \" HeatWave Lakehouse
        "distance": 0.0863,
        "document_name": ""
    },
    {
        "segment": "The Lakehouse feature of HeatWave enables query processing on data in Object Storag
        "distance": 0.1028,
        "document_name": ""
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`", "`demo_db`.`demo_e5_table`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.1028}}
*************************** 2. row ***************************
    id: 2
Output: {"text": "\nHeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine le
"error": null,
"citations": [
    {
```

```
        "segment": "|  HeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machine lea
        "distance": 0.0561,
        "document_name": ""
    },
    {
        "segment": "HeatWave shapes and scaling, and all HeatWave AutoML makes it easy to use machine learn
        "distance": 0.0573,
        "document_name": ""
    },
    {
        "segment": "HeatWave AutoML makes it easy to use machine learning, whether you are a novice user or
        "distance": 0.0598,
        "document_name": ""
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`", "`demo_db`.`demo_e5_table`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.0598}}
*************************** 3. row ***************************
    id: 3
Output: {"text": "\nHeatWave GenAI is a feature of HeatWave that enables natural language communication wit
"error": null,
"citations": [
    {
        "segment": "4.1 HeatWave GenAI Overview HeatWave GenAI is a feature of HeatWave that lets you commu
        "distance": 0.0521,
        "document_name": ""
    },
    {
        "segment": "Chapter 3, HeatWave AutoML. 1.4 HeatWave GenAI The HeatWave GenAI feature of HeatWave l
        "distance": 0.0735,
        "document_name": ""
    },
    {
        "segment": "HeatWave Chat also provides a graphical interface integrated with the Visual Studio Cod
        "distance": 0.0781,
        "document_name": ""
    }
],
"vector_store": ["`demo_db`.`demo_embeddings`", "`demo_db`.`demo_e5_table`"],
"retrieval_info": {"method": "n_citations", "threshold": 0.0781}}
```

The output table generated using the `ML_RAG_TABLE` routine contains an additional details for error reporting. In case the routine fails to generate output for specific rows, details of the errors encountered and default values used are added for the rows in the output column.

Cleaning Up

If you created a new database for testing the steps in this topic, delete the database to free up space:

```
mysql> DROP DATABASE demo_db;
```

What's Next

Learn how to Start a Conversational Chat.

5.9 Starting a Conversational Chat

You can use GenAI Chat to simulate human-like conversations where you can get responses for multiple queries in the same session. GenAI Chat is a conversational agent that utilizes large language models (LLMs) to understand inputs and responds in natural manner. It extends the text generation by using a chat history that lets you ask follow-up questions, and uses the vector search functionality to draw its knowledge from the inbuilt vector store. The responses generated by GenAI Chat are quick and secure as all the communication and processing happens within MySQL AI service.

The sections in this topic describe how to run and manage GenAI Chat.

5.9.1 Running GenAI Chat

When you run GenAI Chat, it automatically loads the `llama3.2-3b-instruct-v1` LLM.

By default, GenAI Chat searches for an answer to a query across all ingested documents by automatically discovering available vector stores, and returns the answer along with relevant citations. You can limit the scope of search to specific document collections available in certain vector stores or specify documents to include in the search.

GenAI Chat also lets you use your own embedding tables for context retrieval. And, it uses only the name of the embedding model used to embed the input query to find relevant tables.

If you do not have a vector store or an embedding table set up, then GenAI Chat uses information available in public data sources to generate a response for your query.

This topic contains the following sections:

- Before You Begin
- Running the Chat
- What's Next

Before You Begin

- Review the GenAI requirements.
- To extend the vector search functionality and ask specific questions about the information available in your proprietary documents stored in the vector store, complete the steps to set up a vector store.

  In this topic, the `HEATWAVE_CHAT` routine uses the vector store table `demo_embeddings` created in the section Ingesting Files into a Vector Store for context retrieval.
- To use your own embedding table for context retrieval, create a table that satisfies the following criteria:
  - The table must contain the following columns:
    - A string column containing the text segments.
    - A vector column containing the vector embeddings of the text segments.
  - A comment on the vector column must specify the name of the embedding model used to generate the vector embeddings.
  - The vector embeddings in your embedding table must be from an embedding model supported by GenAI. To view the list of available embedding models, see In-Database Embedding Model.

    Following is an example of a valid embedding table that can be used for context retrieval:

```
mysql> CREATE TABLE demo_table (id INT AUTO_INCREMENT, demo_text TEXT, primary key (id));
mysql> INSERT INTO demo_table (demo_text) VALUES('What is MySQL?');
mysql> INSERT INTO demo_table (demo_text) VALUES('What is HeatWave?');
mysql> INSERT INTO demo_table (demo_text) VALUES('What is HeatWave GenAI?');
mysql> CALL sys.ML_EMBED_TABLE('demo_schema.demo_table.demo_text', 'demo_schema.demo_table.demo_embedd
JSON_OBJECT('model_id', 'all_minilm_l12_v2'));
```

To learn how to generate vector embeddings and embedding tables, see Generating Vector Embeddings.

If you want to use both inbuilt vector store tables and your own embedding tables for context retrieval, your embedding table must satisfy the following additional requirements:

- Since the inbuilt vector store tables, use predefined column names, the column names in your embedding tables must match the predefined inbuilt vector store table column names as given below:
  - `segment`: name of the mandatory string column containing the text segments.
  - `segment_embedding`: name of the mandatory vector column containing the vector embeddings of the text segments.
  - `document_name`: name of the optional column containing the document names. This column can be of any data type supported by MySQL.
  - `document_id`: name of the optional integer column containing the document IDs.
  - `metadata`: name of the optional JSON column containing metadata for the table.
  - `segment_number`: name of the optional integer column containing segment number.
- The vector embeddings in your embedding table must be from the same embedding model as the vector store table.

Running the Chat

To run GenAI Chat, perform the following steps:

1. Optionally, to speed up vector processing, load the vector store or embedding tables that you want use with GenAI Chat in MySQL AI Engine:

   ```
   mysql> ALTER TABLE TableName SECONDARY_LOAD;
   ```

   Replace *`TableName`* with the name of the vector store table.

   For example:

   ```
   mysql> ALTER TABLE demo_db.demo_embeddings SECONDARY_LOAD;
   ```

   This accelerates processing of vector distance function used to compare vector embeddings and generate relevant output later in this section.

2. To delete previous chat output and state, if any, reset the `@chat_options` variable:

   ```
   mysql> SET @chat_options=NULL;
   ```

   Note

   Ensure that you use the name `@chat_options` for the variable. The `HEATWAVE_CHAT` routine reserves this variable for specifying and saving various chat parameter settings.

3. Optionally, set the `@chat_options` variable in the following scenarios:

   - To use a language other than English, set the `language` model option:

```
mysql> SET @chat_options = JSON_OBJECT("model_options", JSON_OBJECT("language", "Language"));
```

Replace *Language* with the two-letter `ISO 639-1` code for the language you want to use. Default language is `en`, which is English. To view the list of supported languages, see Languages.

For example, to use French set `language` to `fr`:

```
mysql> SET @chat_options = JSON_OBJECT("model_options", JSON_OBJECT("language", "fr"));
```

This resets the `@chat_options` variable, and specifies the language for the chat.

- To use your own embedding tables for context retrieval, change the column names used by the `HEATWAVE_CHAT` routine to filter tables by setting the `vector_store_columns` parameter:

```
mysql> SET @chat_options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "TextSegmentColumnName", "segment_embedding", "Vecto
  "embed_model_id", "EmbeddingModelName"
);
```

Replace the following:

- *TextSegmentColumnName*: the name of the embedding table column that contains the text segments in natural language. If multiple tables contain a string column with the same name, they are all used for context retrieval. Default value is `segment`.
- *VectorEmbeddingColumnName*: the name of the embedding table column that contains vector embeddings of the natural-language text segments. If multiple tables contain a vector column with the same name which contain embeddings from the specified embedding model, they are all used for context retrieval. Default value is `segment_embedding`.
- *EmbeddingModelName*: the name of the embedding model to use to generate the vector embeddings for the input query. The routine uses this embedding model name to find tables generated using the same model for context retrieval. By default, the routine uses `minilm` if the output language is set to English and `multilingual-e5-small` if the output language is set to a language other than English.

By default, the routine uses all the predefined vector store column names to filter tables for context retrieval.

For example:

```
mysql> SET @chat_options = JSON_OBJECT(
  "vector_store_columns", JSON_OBJECT("segment", "demo_text", "segment_embedding", "demo_embeddings")
  "embed_model_id", "all_minilm_l12_v2"
);
```

This resets the `@chat_options` variable to specify the column names used for filtering tables for context retrieval. In this example, all embedding tables containing a string column `demo_text` and a vector column `demo_embeddings` which contains vector embeddings from `all_minilm_l12_v2` are used for context retrieval.

However, since the inbuilt vector store tables use predefined column names, if you change a column name used for filtering tables to any value other than the default value, the inbuilt vector store tables are filtered out and are not used for context retrieval.

4. Then, add your query to GenAI Chat by using the HEATWAVE_CHAT routine:

```
CALL sys.HEATWAVE_CHAT("YourQuery");
```

For example:

```
mysql> CALL sys.HEATWAVE_CHAT("What is HeatWave AutoML?");
```

The output looks similar to the following:

```
|
HeatWave AutoML is an automated machine learning (AutoML) platform that uses a combination of human-in-the-

Here's a brief overview:

**Key Features:**

1. **Automated Model Selection**: HeatWave AutoML allows users to select the best-performing model for thei
2. **Hyperparameter Tuning**: The platform automatically tunes hyperparameters for the selected model, ensu
3. **Data Preprocessing**: HeatWave handles data preprocessing tasks such as feature engineering, normaliza
4. **Model Training**: The platform trains the selected model on the user's dataset and provides real-time
5. **Model Deployment**: Once a model is trained, HeatWave AutoML deploys it to a cloud-based environment f

**Benefits:**

1. **Reduced Time-to-Insight**: Automates the entire machine learning workflow, saving users time and effor
2. **Improved Model Performance**: HeatWave's automated process ensures that models are optimized for perfo
3. **Increased Collaboration |
```

Repeat this step to ask follow-up questions using the HEATWAVE_CHAT routine:

```
mysql> CALL sys.HEATWAVE_CHAT("What learning algorithms does it use?");
```

The output looks similar to the following:

```
|
HeatWave is an AutoML (Automated Machine Learning) platform that uses a combination of various machine lear

HeatWave is built on top of several popular open-source libraries and frameworks, including:

1. **Scikit-learn**: A widely-used Python library for machine learning that provides a variety of algorithm
2. **TensorFlow**: An open-source machine learning framework developed by Google that provides tools for bu
3. **PyTorch**: Another popular open-source machine learning framework that provides a dynamic computation

Some common machine learning algorithms used in AutoML platforms like HeatWave include:

1. **Random Forest**: An ensemble method that combines multiple decision trees to improve the accuracy and
2. **Gradient Boosting**: A type of ensemble method that uses gradient descent to optimize the weights of i
3. **Support Vector Machines (SVMs)**: A supervised learning algorithm |
```

What's Next

Learn how to View Chat Session Details.

5.9.2 Viewing Chat Session Details

This topic describes how to view a chat session details. It contains the following sections:

- Before You Begin
- Viewing Details
- What's Next

Before You Begin

- Complete the steps to run GenAI Chat.

Viewing Details

To view the chat session details, inspect the `@chat_options` variable:

```
mysql> SELECT JSON_PRETTY(@chat_options);
```

The output includes the following details about a chat session:

- *Vector store tables:* in the database which were referenced by GenAI Chat.
- *Text segments:* that were retrieved from the vector store and used as context to prepare responses for your queries.
- *Chat history:* which includes both your queries and responses generated by GenAI Chat.
- *LLM details:* which was used by the routine to generate the responses.

The output looks similar to the following:

```
| {
  "tables": [
    {
      "table_name": "`demo_embeddings`",
      "schema_name": "`demo_db`"
    }
  ],
  "response": "\nThe output of the follow-up question is:\n| HeatWave AutoML uses a variety of machine lear
  "documents": [
    {
      "id": "/export/home/tmp/mysql-files/demo-directory/heatwave-en.pdf",
      "title": "heatwave-en.pdf",
      "segment": "Repeat this step to ask follow-up questions using the HEATWAVE_CHAT routine:\nCALL sys.HE
      "distance": 0.0622
    },
    {
      "id": "/export/home/tmp/mysql-files/demo-directory/heatwave-en.pdf",
      "title": "heatwave-en.pdf",
      "segment": "HeatWave AutoML makes it easy to use machine learning, whether you are a novice user or a
      "distance": 0.0646
    },
    {
      "id": "/export/home/tmp/mysql-files/demo-directory/heatwave-en.pdf",
      "title": "heatwave-en.pdf",
      "segment": "HeatWave shapes and scaling, and all HeatWave AutoML makes it easy to use machine learnin
      "distance": 0.0679
    }
  ],
  "chat_history": [
    {
      "user_message": "What is HeatWave AutoML?",
      "chat_query_id": "7aa7824c-5d8a-11f0-a2c5-020017192be1",
      "chat_bot_message": "\nHeatWave AutoML is a feature of MySQL HeatWave that makes it easy to use machi
    },
    {
      "user_message": "What learning algorithms does it use?",
      "chat_query_id": "93730281-5d8a-11f0-a2c5-020017192be1",
      "chat_bot_message": "\nThe output of the follow-up question is:\n| HeatWave AutoML uses a variety of
    }
  ],
  "model_options": {
```

```
    "model_id": "llama3.2-3b-instruct-v1"
  },
  "request_completed": true
} |
```

What's Next

See Chapter 7, *MySQL AI Routines*.

Chapter 6 Review Performance and Usage

Table of Contents

MySQL AI lets you offload vector-based tables to the MySQL AI Engine for accelerated processing of queries that uses at least one of the vector functions. The MySQL Performance Schema collects statistics on the usage of the AI engine and different functions that you perform with MySQL AI.

Use SQL queries to access this data and check the system status and performance.

6.1 MySQL AI Performance Schema Tables

MySQL AI Performance Schema tables provide information about AI nodes, and about tables and columns that are currently loaded in the MySQL AI Engine (AI engine).

6.1.1 The rpd_column_id Table

The `rpd_column_id` table provides information about columns of tables that are loaded in the MySQL AI Engine.

The `rpd_column_id` table has these columns:

- `ID`

  A unique identifier for the column.

- `TABLE_ID`

  The ID of the table to which the column belongs.

- `COLUMN_NAME`

  The column name.

The `rpd_column_id` table is read-only.

6.1.2 The rpd_columns Table

The `rpd_columns` table provides column encoding information for columns of tables loaded in the MySQL AI Engine.

The `rpd_columns` table has these columns:

- `TABLE_ID`

A unique identifier for the table.

- `COLUMN_ID`

  A unique identifier for the table column.

- `NDV`

  The number of distinct values in the column.

- `ENCODING`

  The type of encoding used.

- `DATA_PLACEMENT_INDEX`

  The data placement key index ID associated with the column. Index value ranges from 1 to 16. NULL indicates that the column is not defined as a data placement key.

- `DICT_SIZE_BTYES`

  The dictionary size per column, in bytes.

The `rpd_columns` table is read-only.

6.1.3 The rpd_ml_stats Table

The `rpd_ml_stats` table tracks the usage of successful MySQL AI routines. These metrics reset whenever the respective DB system restarts.

The following AutoML routines are tracked:

- `ML_TRAIN`
- `ML_EXPLAIN`
- `ML_PREDICT_ROW`
- `ML_PREDICT_TABLE`
- `ML_EXPLAIN_ROW`
- `ML_EXPLAIN_TABLE`

The following GenAI routines are tracked:

- `ML_GENERATE`
- `ML_EMBED_ROW`

The `rpd_ml_stats` table has these columns:

- `STATUS_NAME`

  Identifies the type of meter tracking usage.

- `STATUS_VALUE`

Displays metrics for metering. Content is displayed in JSON format.

Metrics in the table are entries as JSON values. The following metrics are used:

- `n_cells`

  The total number of table cells processed by the AutoML routine for all invocations.

- `n_cells_user_excluded`

  The total number of table cells manually excluded for the AutoML routine.

- `n_blob_cells`

  The total number of table BLOB cells processed by the AutoML routine for all invocations.

- `table_size_bytes`

  The total number of bytes of data processed by the AutoML routine for all invocations.

- `blob_size_bytes`

  The total number of bytes of BLOB/TEXT data processed by the AutoML routine for all invocations.

- `model_size_bytes`

  The total number of bytes of data for the AutoML model that is trained. This includes any explainer models. This metric only applies to the `ML_TRAIN` and `ML_EXPLAIN` AutoML routines. All other routines will display NULL values.

- `input_size_bytes`

  The cumulative size in bytes of all input string/document invocations ingested by the GenAI routine.

- `context_size_bytes`

  The size in bytes of the context string referenced when generating the response. This metric only applies to the `ML_GENERATE` GenAI routine since the `ML_EMBED_ROW` routine does not have context. The metric will still appear for `ML_EMBED_ROW`, but will display a value of 0.

- `output_size_bytes`

  The cumulative size in bytes of responses generated by all invocations for the GenAI routine.

- `n_invocations`

  The total number of times the routine has been successfully invoked on the MySQL AI Engine.

- `last_updated_timestamp`

  The POSIX timestamp of the last call.

6.1.4 The rpd_nodes Table

The `rpd_nodes` table provides information about AI nodes.

The `rpd_nodes` table has these columns:

- `ID`

  A unique identifier for the MySQL AI Engine (AI engine).

- `CORES`

  The number of cores used by the AI engine.

- `MEMORY_USAGE`

  Node memory usage in bytes. The value is refreshed every four seconds. If a query starts and finishes in the four seconds between refreshes, the memory used by the query is not accounted for in the reported value.

- `MEMORY_TOTAL`

  The total memory in bytes allocated to the AI engine.

- `BASEREL_MEMORY_USAGE`

  The base relation memory footprint per node.

- `STATUS`

  The status of the AI engine. Possible statuses include:

  - `NOTAVAIL_RNSTATE`

    Not available.

  - `AVAIL_RNSTATE`

    Available.

  - `DOWN_RNSTATE`

    Down.

  - `DEAD_RNSTATE`

    The node is not operational.

- `IP`

  IP address of the AI engine.

- `PORT`

  The port on which the AI engine was started.

- `CLUSTER_EVENT_NUM`

  The number of cluster events such as node down, node up, and so on.

- `NUM_OBJSTORE_GETS`

  Number of `GET` requests from the AI engine to the disk.

- `NUM_OBJSTORE_PUTS`

The number of PUT requests from the AI engine to the disk.

- NUM_OBJSTORE_DELETES

  The number of DELETE requests from the AI engine to the disk.

- ML_STATUS

  AutoML status. Possible status values include:

  - UNAVAIL_MLSTATE: AutoML is not available.
  - AVAIL_MLSTATE: AutoML is available.
  - DOWN_MLSTATE: AutoML declares the node is down.

The rpd_nodes table is read-only.

The rpd_nodes table may not show the current status for a new node or newly configured node immediately. The rpd_nodes table is updated after the node has successfully joined the cluster.

If additional nodes fail while node recovery is in progress, the newly failed nodes are not detected and their status is not updated in the performance_schema.rpd_nodes table until after the current recovery operation finishes and the nodes that failed previously have rejoined the cluster.

6.1.5 The rpd_table_id Table

The rpd_table_id table provides the ID, name, and schema of the tables loaded in the MySQL AI Engine.

The rpd_table_id table has these columns:

- ID

  A unique identifier for the table.

- NAME

  The full table name including the schema.

- SCHEMA_NAME

  The schema name.

- TABLE_NAME

  The table name.

The rpd_table_id table is read-only.

6.1.6 The rpd_tables Table

The rpd_tables table provides the system change number (SCN) and load pool type for tables loaded in the MySQL AI Engine (AI engine).

The rpd_tables table has these columns:

- `ID`

  A unique identifier for the table.

- `SNAPSHOT_SCN`

  The system change number (SCN) of the table snapshot. The SCN is an internal number that represents a point in time according to the system logical clock that the table snapshot was transactionally consistent with the source table.

- `PERSISTED_SCN`

  The SCN up to which changes are persisted.

- `POOL_TYPE`

  The load pool type of the table. Possible values are `SNAPSHOT` and `TRANSACTIONAL`.

- `DATA_PLACEMENT_TYPE`

  The data placement type.

- `NROWS`

  The number of rows that are loaded for the table. The value is set initially when the table is loaded, and updated as changes are propagated.

- `LOAD_STATUS`

  The load status of the table. Statuses include:

  - `NOLOAD_RPDGSTABSTATE`

    The table is not yet loaded.

  - `LOADING_RPDGSTABSTATE`

    The table is being loaded.

  - `AVAIL_RPDGSTABSTATE`

    The table is loaded and available for queries.

  - `UNLOADING_RPDGSTABSTATE`

    The table is being unloaded.

  - `INRECOVERY_RPDGSTABSTATE`

    The table is being recovered. After completion of the recovery operation, the table is placed back in the `UNAVAIL_RPDGSTABSTATE` state if there are pending recoveries.

  - `STALE_RPDGSTABSTATE`

    A failure during change propagation, and the table has become stale.

  - `UNAVAIL_RPDGSTABSTATE`

    The table is unavailable.

- LOAD_PROGRESS

  The load progress of the table expressed as a percentage value.

- SIZE_BYTES

  The amount of data loaded for the table, in bytes.

- NROWS:

  The number of rows loaded to the external table.

- QUERY_COUNT

  The number of queries that referenced the table.

- LAST_QUERIED

  The timestamp of the last query that referenced the table.

- LOAD_START_TIMESTAMP

  The load start timestamp for the table.

- LOAD_END_TIMESTAMP

  The load completion timestamp for the table.

- RECOVERY_SOURCE

  Indicates the source of the last successful recovery for a table.

- RECOVERY_START_TIMESTAMP

  The timestamp when the latest successful recovery started.

- RECOVERY_END_TIMESTAMP

  The timestamp when the latest successful recovery ended.

The rpd_tables table is read-only.

6.2 Option Tracker

The Option Tracker component provides usage information about different features and components of MySQL AI.

For more information, see Option Tracker Component.

The integer flag usedCounter is incremented in real-time and persisted to storage every hour.

- The option_tracker_usage_get() function returns a value similar to the following:

```
mysql> SELECT option_tracker_usage_get('Berry Picker');
+---------------------------------------------------+
| option_tracker_usage_get('Berry Picker')          |
+---------------------------------------------------+
| {"usedCounter": 30, "usedDate": "2025-14-16T09:14:41Z"} |
+---------------------------------------------------+
```

- The `option_tracker_usage_set()` function accepts JSON-formatted string similar to the following for the `usage_data` argument:

```
{
  "usedCounter": "integer"
  "usedDate": "ISO8601 date"
}
```

Chapter 7 MySQL AI Routines

Table of Contents

The sections in this chapter list and describe various routines available in MySQL AI.

7.1 AutoML Routines

MySQL AI AutoML routines reside in the MySQL `sys` schema.

7.1.1 ML_TRAIN

Run the `ML_TRAIN` routine on a training dataset to produce a trained machine learning model.

Before training models, make sure to review the following:

- Additional MySQL HeatWave AutoML Requirements
- Supported Data Types for MySQL HeatWave AutoML
- Train a Model
- Machine Learning Use Cases

This topic has the following sections. Refer to the appropriate sections depending on the type of machine learning model you would like to train.

- ML_TRAIN Syntax

ML_TRAIN Syntax

```
mysql> CALL sys.ML_TRAIN ('table_name', 'target_column_name', [options | NULL], model_handle);

options: {
     JSON_OBJECT("key","value"[,"key","value"] ...)
          "key","value": {
          ['task', {'classification'|'regression'|'forecasting'|'anomaly_detection'|'log_anomaly_detection'|'r
          ['datetime_index', 'column']
          ['endogenous_variables', JSON_ARRAY('column'[,'column'] ...)]
          ['exogenous_variables', JSON_ARRAY('column'[,'column'] ...)]
          ['model_list', JSON_ARRAY('model'[,'model'] ...)]
          ['exclude_model_list', JSON_ARRAY('model'[,'model'] ...)]
          ['optimization_metric', 'metric']
          ['include_column_list', JSON_ARRAY('column'[,'column'] ...)]
          ['exclude_column_list', JSON_ARRAY('column'[,'column'] ...)]
          ['contamination',  'contamination factor']
          ['supervised_submodel_options', {'n_neighbors', 'N', 'min_labels', N}']
          ['ensemble_score', 'ensemble metric']
          ['users', 'users_column']
          ['items', 'items_column']
          ['notes', 'notes_text']
          ['feedback', {'explicit'  ['implicit'}]
          ['feedback_threshold', 'threshold']
          ['item_metadata', JSON_OBJECT('table_name'[,'database_name.table_name'] ...)]
          ['document_column', 'column_name']
          ['logad_options', JSON_OBJECT(("key","value"[,"key","value"] ...)
                 "key","value": {
                              ['additional_masking_regex', JSON_ARRAY('regular_expression'[,'regular_expressio
                              ['window_size', 'N']
```

```
                              ['window_stride', 'N']
                              ['log_source_column', 'column']
                              }
          }
}
```

Required ML_TRAIN Parameters

Set the following parameters to train all machine learning models.

- `table_name`: The name of the table that contains the labeled training dataset. The table name must be valid and fully qualified, so it must include the database name, *`database_name.table_name`*. The table cannot exceed 10 GB, 100 million rows, or 1017 columns.
- `target_column_name`: The name of the target column containing ground truth values.

  MySQL HeatWave AutoML does not support a text target column.

  If training an unsupervised Anomaly detection model (unlabeled data), set `target_column_name` to `NULL`.

  Forecasting does not require `target_column_name`, and it can be set to `NULL`.
- `model_handle`: A user-defined session variable that stores the machine learning model handle for the duration of the connection. User variables are written as *`@var_name`*. Any valid name for a user-defined variable is permitted. For example, `@my_model`.

  If you set a value to the `model_handle` variable before calling `ML_TRAIN`, that model handle is used for the model. A model handle must be unique in the model catalog. We recommend this method.

  If you don't set a value to the `model_handle` variable, MySQL HeatWave AutoML generates one. When `ML_TRAIN` finishes executing, retrieve the generated model handle by querying the session variable. See Model Handles to learn more.

Common ML_TRAIN Options

The following optional parameters apply to more than one type of machine learning task. They are specified as key-value pairs in `JSON` format. If an option is not specified, the default setting is used. If no options are specified, you can specify `NULL` in place of the `JSON` argument.

- `task`: Specifies the machine learning task.
  - `classification`: The default value if a task is not set. Use this task type to assign items to defined categories.
  - `regression`: Use this task type if the target column is a continuous numerical value. This task generates predictions based on the relationship between a dependent variable and one or more independent variables.
  - `forecasting`: Use this task type if you have a date-time column that requires a timeseries forecast. To use this task, you must set a target column, the date-time column (`datetime_index`), and endogenous variables (`endogenous_variables`).
  - `anomaly_detection`: Use this task type to detect unusual patterns in data.
  - `log_anomaly_detection`: Use this task to detect unusual patterns in log data.
  - `recommendation`: Use this task type for generate recommendations for users and items.

  - topic_modeling: Use this task to cluster word groups and similar expressions that best characterize the documents.

- model_list: The type of model to be trained. If more than one model is specified, the best model type is selected from the list. See Model Types.

  This option cannot be used together with the exclude_model_list option.

- exclude_model_list: Model types that should not be trained. Specified model types are excluded from consideration during model selection. See Model Types.

  This option cannot be specified together with the model_list option.

- optimization_metric: The scoring metric to optimize for when training a machine learning model. The metric must be compatible with the task type and the target data. See Section 7.1.14, "Optimization and Scoring Metrics".

  This is not supported for anomaly_detection tasks. Instead, metrics for anomaly detection can only be used with the ML_SCORE routine.

- include_column_list: ML_TRAIN must include this list of columns.

  For classification, regression, anomaly_detection and recommendation tasks, include_column_list ensures that ML_TRAIN will not drop these columns.

  For forecasting tasks, include_column_list can only include exogenous_variables. If include_column_list is included in the ML_TRAIN options for a forecasting task with at least one exogenous_variables, this forces ML_TRAIN to only consider those models that support exogenous_variables.

  All columns in include_column_list must be included in the training table.

- exclude_column_list: Feature columns of the training dataset to exclude from consideration when training a model. Columns that are excluded using exclude_column_list do not also need to be excluded from the dataset used for predictions.

  The exclude_column_list cannot contain any columns provided in endogenous_variables, exogenous_variables, and include_column_list.

- notes: Add notes to the model_metadata for your own reference.

Refer to the following model-specific parameters to train different types of machine learning models.

Parameters to Train a Classification Model

To train a classification model, set the task to classification.

If the task is set to NULL, or if all training options is set to NULL, a classification model is trained by default.

Syntax Examples for Classification Training

- The following example sets the model handle before training, which is good practice. See Defining Model Handle. The task is set to classification.

```
mysql> SET @census_model = 'census_manual';
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @cens
```

- The following example sets all options to `NULL`, so `ML_TRAIN` runs the `classification` task option by default.

```
mysql> CALL sys.ML_TRAIN('census_data.census_train', 'revenue', NULL, @census_model);
```

Parameters to Train a Regression Model

To train a regression model, set the `task` to `regression`.

Syntax Examples for Regression Training

- The following example specifies the `regression` task type.

```
mysql> CALL sys.ML_TRAIN('nyc_taxi.nyc_taxi_train', 'tip_amount', JSON_OBJECT('task', 'regression'), @ny
```

Parameters to Train a Forecasting Model

See the following to learn more about forecasting models:

- Forecasting Task Types
- Prediction Intervals
- Train a Forecasting Model

To train a forecasting model, set the `task` to `forecasting` and set the following required parameters.

- `datetime_index`: The column name for a datetime column that acts as an index for the forecast variable. The column can be one of the supported datetime column types, `DATETIME`, `TIMESTAMP`, `DATE`, `TIME`, and `YEAR`, or an auto-incrementing index.

  The forecast models `SARIMAXForecaster`, `VARMAXForecaster`, and `DynFactorForecaster` cannot back test, that is forecast into training data, when using `exogenous_variables`. Therefore, the predict table must not overlap the `datetime_index` with the training table. The start date in the predict table must be a date immediately following the last date in the training table when `exogenous_variables` are used. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023.

  The `datetime_index` for the predict table must not have missing dates after the last date in the training table. For example, the predict table has to start with year 2024 if the training table with `YEAR` data type `datetime_index` ends with year 2023. The predict table cannot start with year, for example, 2025 or 2030, because that would miss out 1 and 6 years, respectively.

  When *`options`* do not include `exogenous_variables` , the predict table can overlap the `datetime_index` with the training table. This supports back testing.

  The valid range of years for `datetime_index` dates must be between 1678 and 2261. It will cause an error if any part of the training table or predict table has dates outside this range. The last date in the training table plus the predict table length must still be inside the valid year range. For example, if the `datetime_index` in the training table has `YEAR` data type, and the last date is year 2023, the predict table length must be less than 238 rows: 2261 minus 2023 equals 238 rows.

- `endogenous_variables`: The column or columns to be forecast.

  Univariate forecasting models support a single numeric column, specified as a `JSON_ARRAY`. This column must also be specified as the `target_column_name`, because that field is required, but it is not used in that location.

Multivariate forecasting models support multiple numeric columns, specified as a `JSON_ARRAY`. One of these columns must also be specified as the `target_column_name`.

`endogenous_variables` cannot be text.

Set the following forecasting options as required to train forecasting models.

- `exogenous_variables`: For forecasting tasks, the column or columns of independent, non-forecast, predictive variables, specified as a `JSON_ARRAY`. These optional variables are not forecast, but help to predict the future values of the forecast variables. These variables affect a model without being affected by it. For example, for sales forecasting these variables might be advertising expenditure, occurrence of promotional events, weather, or holidays.

  `ML_TRAIN` will consider all supported models during the algorithm selection stage if *`options`* includes `exogenous_variables`, including models that do not support `exogenous_variables`.

  For example, if *`options`* includes univariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` will consider `NaiveForecaster`, `ThetaForecaster`, `ExpSmoothForecaster`, `ETSForecaster`, `STLwESForecaster`, `STLwARIMAForecaster`, and `SARIMAXForecaster`. `ML_TRAIN` will ignore `exogenous_variables` if the model does not support them.

  Similarly, if *`options`* includes multivariate `endogenous_variables` with `exogenous_variables`, then `ML_TRAIN` will consider `VARMAXForecaster` and `DynFactorForecaster`.

  If *`options`* also includes `include_column_list`, this will force `ML_TRAIN` to only consider those models that support `exogenous_variables`.

- `include_column_list`: Can only include `exogenous_variables`. If `include_column_list` contains at least one `exogenous_variables`, this will force `ML_TRAIN` to only consider those models that support `exogenous_variables`.

Syntax Examples for Forecast Training

- The following example specifies the `forecasting` task type, and the additional required parameters, `datetime_index` and `endogenous_variables`.

```
mysql> CALL sys.ML_TRAIN('ml_data.opsd_germany_daily_train', 'consumption', JSON_OBJECT('task', 'forecasting
```

- The following example specifies the `OrbitForecaster` forecasting model with exogenous variables.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.opsd_germany_daily_train', NULL, JSON_OBJECT('task', 'forecasting', 'date
       JSON_ARRAY('consumption'), 'exogenous_variables', JSON_ARRAY('wind', 'solar', 'wind_solar'), 'model_l
```

- The following example specifies the `OrbitForecaster` forecasting model without exogenous variables.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`datetime_train`', 'C1', JSON_OBJECT('task', 'forecasting', 'datetime_ind
       JSON_ARRAY('C1'), 'model_list', JSON_ARRAY('OrbitForecaster')), @datetime_model);
```

Parameters to Train an Anomaly Detection Model

See the following to learn more about anomaly detection models:

- Anomaly Detection Model Types
- Anomaly Detection Learning Types
- Anomaly Detection for Logs

To train an anomaly detection model, set the appropriate required parameters depending on the type of anomaly detection model to train.

- Set the `task` parameter to `anomaly_detection` for running anomaly detection on table data, or `log_anomaly_detection` for running anomaly detection on log data.
- If running an unsupervised model, the `target_column_name` parameter must be set to `NULL`.
- If running a semi-supervised model:
  - The `target_column_name` parameter must specify a column whose only allowed values are 0 (normal), 1 (anomalous), and NULL (unlabeled). All rows will be used to train the unsupervised component, while the rows with a value different than NULL will be used to train the supervised component.
  - The `experimental` option must be set to `semisupervised`.
- If running anomaly detection on log data, the input table can only have the following columns:
  - The column containing the logs.
  - If including logs from different sources, a column containing the source of each log. Identify this column with the `log_source_column` option.
  - If including labeled data, a column identifying the labeled log lines. See Semi-supervised Anomaly Detection to learn more.
  - At least one column must act as the primary key to establish the temporal order of logs. If the primary key column (or columns) is not one of the previous required columns (log data, source of log, or label), then you must use the `exclude_column_list` option when running `ML_TRAIN` to exclude all primary key columns that don't include required data. See Syntax Examples for Anomaly Detection Training to review relevant examples.
  - If the input table has additional columns to the ones permitted, you must use the `exclude_column_list` option to exclude irrelevant columns.

Set the following options as needed for anomaly detection models:

- `contamination`: Represents an estimate of the percentage of outliers in the training table.
  - The contamination factor is calculated as: estimated number of rows with anomalies/total number of rows in the training table.
  - The contamination value must be greater than 0 and less than 0.5. The default value is 0.01.
- `model_list`: You can select the Principal Component Analysis (PCA), Generalized Local Outlier Factor (GLOF), or Generalized kth Nearest Neighbors (GkNN) model. If no option is specified, the default model is GkNN. Selecting more than one model or an unsupported model produces an error.

To train a semi-supervised anomaly detection model, set the following options:

- `supervised_submodel_options`: Allows you to set optional override parameters for the supervised model component. The only model supported is `DistanceWeightedKNNClassifier`. The following parameters are supported:
  - `n_neighbors`: Sets the desired k value that checks the k closest neighbors for each unclassified point. The default value is 5 and the value must be an integer greater than 0.

  - `min_labels`: Sets the minimum number of labeled data points required to train the supervised component. If fewer labeled data points are provided during training of the model, `ML_TRAIN` fails. The default value is 20 and the value must be an integer greater than 0.

- `ensemble_score`: This option specifies the metric to use to score the ensemble of unsupervised and supervised components. It identifies the optimal weight between the two components based on the metric. The supported metrics are `accuracy`, `precision`, `recall`, and `f1`. The default metric is `f1`.

To train a model for anomaly detection on log data, set the following options:

- `logad_options`: A `JSON_OBJECT` that allows you to configure the following options.

  - `additional_masking_regex`: Allows you to mask log data in a `JSON_ARRAY`. By default, the following parameters are automatically masked during training.

    - IP
    - DATETIME
    - TIME
    - HEX
    - IPPORT
    - OCID

  - `window_size`: Specifies the maximum number of log lines to be grouped for anomaly detection. The default value is 10.

  - `window_stride`: Specifies the stride value to use for segmenting log lines. For example, there is log A, B, C, D, and E. The `window_size` is 3, and the `window_stride` is 2. The first row has log A, B, and C. The second row has log C, D, and E. If this value is equal to `window_size`, there is no overlapping of log segments. The default value is 3.

  - `log_source_column`: Specifies the column name that contains the source identifier of the respective log lines. Log lines are grouped according to their respective source (for example, logs from multiple MySQL databases that are in the same table). By default, all log lines are assumed to be from the same source.

Anomaly detection models don't support the following options during training:

- `exclude_model_list`
- `optimization_metric`

Syntax Examples for Anomaly Detection Training

- The following example specifies the `anomaly_detection` task type.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', NULL, JSON_OBJECT('task',
Query OK, 0 rows affected (46.59 sec)
```

- The following example specifies the `anomaly_detection` task with a `contamination` option. Query the model catalog metadata to check the value of the `contamination` option.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', NULL,
       JSON_OBJECT('task', 'anomaly_detection', 'contamination', 0.013, 'exclude_column_list',
```

```
        JSON_ARRAY('target')), @anomaly_with_contamination);
Query OK, 0 rows affected (50.22 sec)

mysql> SELECT JSON_EXTRACT(model_metadata, '$.contamination') FROM ML_SCHEMA_root.MODEL_CATALOG WHERE mod
+-------------------------------------------------+
| JSON_EXTRACT(model_metadata, '$.contamination') |
+-------------------------------------------------+
| 0.013000000268220901                            |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

- The following example enables semi-supervised learning using all defaults. The *target_column_name* is set to target. The experimental option is set to semisupervised.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.anomaly_train_with_partial_target', "target", CAST('{"task": "anomaly_
```

- The following example enables semi-supervised learning with additional options.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`anomaly_train_with_partial_target`', "target", CAST('{"task": "anoma
       {"supervised_submodel_options": {""min_labels": 10, "n_neighbors": 3}, "ensemble_score": "recall"
```

  Where:

  - The supervised_submodel_options parameter min_labels is set to 10.
  - The supervised_submodel_options parameter n_neighbors is set to 3.
  - The ensemble_score option is set to the recall metric.

- The following example selects the PCA (Principal Component Analysis) anomaly detection model.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_anomaly_detection_v1.`volcanoes-b3_anomaly_train`', NULL,
       JSON_OBJECT('task', 'anomaly_detection', 'exclude_column_list',
       JSON_ARRAY('target'), 'model_list', JSON_ARRAY('PCA')), @anomaly_pca);
```

- The following example selects the GLOF (Generalized Local Outlier Factor) anomaly detection model.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_anomaly_detection_v1.`volcanoes-b3_anomaly_train`', NULL,
       JSON_OBJECT('task', 'anomaly_detection', 'exclude_column_list',
       JSON_ARRAY('target'), 'model_list', JSON_ARRAY('GLOF')), @anomaly_glof);
```

- The following example does not specify an algorithm model for the model_list option. If no model is specified, the default model GkNN is used.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_anomaly_detection_v1.`volcanoes-b3_anomaly_train`', NULL,
       JSON_OBJECT('task', 'anomaly_detection', 'exclude_column_list',
       JSON_ARRAY('target'), 'model_list', JSON_ARRAY()), @anomaly_empty_list);
```

- The following example runs the log_anomaly_detection task with available default values.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_just_patterns`', NULL, JSON_OBJECT('task', 'log_anomaly_
```

- The following example runs the log_anomaly_detection task with the PCA anomaly detection model.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_just_patterns`', NULL,
       JSON_OBJECT('task', 'log_anomaly_detection', 'model_list', JSON_ARRAY('PCA')), @logad_model);
```

- An ML_TRAIN example that excludes two primary key columns: primary_key_column1 and primary_key_column2. These columns must be excluded because they do not have one of the required items of data for training: the log data, the source of the log, or the label.

```
mysql>CALL sys.ML_TRAIN( 'mlcorpus.log_anomaly_two_primary', NULL,
```

```
        JSON_OBJECT( 'task', 'log_anomaly_detection', 'logad_options',
        JSON_OBJECT('window_size', 2, 'window_stride', 1), 'exclude_column_list',
        JSON_ARRAY('primary_key_column1', 'primary_key_column2') ), @log_anomaly_us );
```

- The following example runs the `log_anomaly_detection` task and masks log data with the `additional_masking_regex` option. In addition to the default parameters that are automatically masked, email addresses from Yahoo, Hotmail, and Gmail are also masked. The `log_source_column` option is also included, which specifies the column that identifies the respective source of the log line.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_sourced`', NULL,
        JSON_OBJECT('task', 'log_anomaly_detection', 'logad_options',
        JSON_OBJECT('additional_masking_regex',
        JSON_ARRAY('(\W|^)[\w.\-]{0,25}@(yahoo|hotmail|gmail)\.com(\W|$)'), 'log_source_column', 'source')),
```

- The following example sets semi-supervised learning for training the log data for anomaly detection. The window size is also set to a value of 4, and the window stride is set to 1.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.`log_anomaly_semi`', "label",
        JSON_OBJECT('task', 'log_anomaly_detection', 'logad_options',
        JSON_OBJECT('window_size', 4, 'window_stride', 1), "experimental",
        JSON_OBJECT("semisupervised", JSON_OBJECT("supervised_submodel_options",
        JSON_OBJECT("min_labels", 10)))), @log_anomaly_us);
```

Parameters to Train a Recommendation Model

See Recommendation Task Types to learn more about recommendation models.

To train a recommendation model, set the `task` to `recommendation` and set the following required parameters.

- `users`: Specifies the column name corresponding to the user ids. Values in this column must be in a `STRING` data type, otherwise an error will be generated during training.

  This must be a valid column name, and it must be different from the `items` column name.

- `items`: Specifies the column name corresponding to the item ids. Values in this column must be in a `STRING` data type, otherwise an error will be generated during training.

  This must be a valid column name, and it must be different from the `users` column name.

To train a recommendation model with explicit feedback, set `feedback` to `explicit`. If `feedback` is not set, the default value is `explicit`.

To train a recommendation model with implicit feedback, set `feedback` to `implicit` and set the following option as needed:

- `feedback_threshold`: The feedback threshold for a recommendation model that uses implicit feedback. It represents the threshold required to be considered positive feedback. For example, if numerical data records the number of times users interact with an item, you might set a threshold with a value of 3. This means users would need to interact with an item more than three times to be considered positive feedback.

To train a content-based recommendation model, set `feedback` to `implicit` and set the following required parameters:

- `item_metadata`: Defines the table that has item description. It is a JSON object that can have the `table_name` option as a key, which specifies the table that has item descriptions. This table must only have two columns: one corresponding to the `item_id`, and the other with a `TEXT` data type (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT) that has the description of the item.

- `table_name`: To be used with the `item_metadata` option. It specifies the table name that has item descriptions. It must be a string in a fully qualified format (database_name.table_name) that specifies the table name.

Syntax Examples for Recommendation Training

- The following example specifies the `SVD` recommendation model type.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.foursquare_NYC_train', 'rating', JSON_OBJECT('task', 'recommendation'
Query OK, 0 rows affected (11.31 sec)

mysql> SELECT model_type FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------+
| model_type |
+------------+
| SVD        |
+------------+
1 row in set (0.00 sec)
```

- The following example specifies the `SVDpp` recommendation model type.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.foursquare_NYC_train', 'rating', JSON_OBJECT('task', 'recommendation'
Query OK, 0 rows affected (13.97 sec)

mysql> SELECT model_type FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------+
| model_type |
+------------+
| SVDpp      |
+------------+
1 row in set (0.00 sec)
```

- The following example specifies the `NMF` recommendation model type.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.foursquare_NYC_train', 'rating', JSON_OBJECT('task', 'recommendation'
Query OK, 0 rows affected (12.28 sec)

mysql> SELECT model_type FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------+
| model_type |
+------------+
| NMF        |
+------------+
1 row in set (0.00 sec)
```

- The following example specifies three models for the `model_list` option. From those three recommendation models, the `SVD` model is automatically selected for training.

```
mysql> SET @allowed_models = JSON_ARRAY('SVD', 'SVDpp', 'NMF');

mysql> CALL sys.ML_TRAIN('mlcorpus.foursquare_NYC_train', 'rating', JSON_OBJECT('task', 'recommendation'
Query OK, 0 rows affected (14.88 sec)

mysql> SELECT model_type FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------+
| model_type |
+------------+
| SVD        |
+------------+
1 row in set (0.00 sec)
```

- The following example specifies five models for the `exclude_model_list` option. The `SVDpp` recommendation model is automatically selected from the remaining available models.

```
mysql> SET @exclude_models= JSON_ARRAY('NormalPredictor', 'Baseline', 'SlopeOne', 'CoClustering', 'SVD');

mysql> CALL sys.ML_TRAIN('mlcorpus.foursquare_NYC_train', 'rating', JSON_OBJECT('task', 'recommendation', 'u
Query OK, 0 rows affected (14.71 sec)

mysql> SELECT model_type FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@model;
+------------+
| model_type |
+------------+
| SVDpp      |
+------------+
1 row in set (0.00 sec)
```

- The following example specifies the `recommendation` task with implicit feedback.

```
mysql> CALL sys.ML_TRAIN('mlcorpus.training_table', 'rating',
          JSON_OBJECT('task', 'recommendation',
          'users', 'user_id',
          'items', 'item_id',
          'feedback', 'implicit'),
          @model);
Query OK, 0 rows affected (2 min 13.6415 sec)
```

- The following example trains a content-based recommendation model by specifying a table with item descriptions (`mlcorpus_recsys.`citeulike_items_sample`). The optimization metric `hit_ratio_at_k` is used. The model must use implicit feedback.

```
mysql> CALL sys.ML_TRAIN('mlcorpus_recsys.`citeulike_train_sample`', 'rating', JSON_OBJECT('task', 'recommen
       JSON_ARRAY('CTR'), 'users', 'user_id', 'items', 'item_id','feedback', 'implicit', 'optimization_metri
```

Parameters to Train a Model with Topic Modeling

To train a machine learning model with topic modeling, set the `task` to `topic_modeling` and set the following required parameter:

- `document_column`: Specify the column name that contains the text to train.

The following parameters are not supported for training machine learning models with topic modeling:

- `model_list`
- `optimization_metric`
- `exclude_model_list`
- `exclude_column_list`
- `include_column_list`

Syntax Examples for Topic Modeling Training

The following example runs the `topic_modeling` task with the required defined parameters.

```
mysql> CALL sys.ML_TRAIN('topic_modeling_data.text_types_train', NULL, JSON_OBJECT('task', 'topic_modeling', '
```

ML_TRAIN and ML_EXPLAIN

The `ML_TRAIN` routine also runs the `ML_EXPLAIN` routine with the default Permutation Importance model for prediction explainers and model explainers. See Generate Model Explanations. To train other prediction explainers and model explainers use the `ML_EXPLAIN` routine with the preferred explainer after `ML_TRAIN`.

ML_EXPLAIN does not support the anomaly_detection and recommendation tasks, and ML_TRAIN does not run ML_EXPLAIN.

Additional Syntax Examples

- The model_list option permits specifying the type of model to be trained. If more than one type of model specified, the best model type is selected from the list. For a list of supported model types, see Model Types. This option cannot be used together with the exclude_model_list option.

  The following example trains either an XGBClassifier or LGBMClassifier model.

```
mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class', JSON_OBJECT('task','classification', 'model_list
```

- The exclude_model_list option specifies types of models that should not be trained. Specified model types are excluded from consideration. For a list of model types you can specify, see Model Types. This option cannot be used together with the model_list option.

  The following example excludes the LogisticRegression and GaussianNB models.

```
mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class', JSON_OBJECT('task','classification', 'exclude_mo
```

- The optimization_metric option specifies a scoring metric to optimize for. See: Optimization and Scoring Metrics.

  The following example optimizes for the neg_log_loss metric.

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task','classification'
```

- The exclude_column_list option specifies feature columns to exclude from consideration when training a model.

  The following example excludes the 'age' column from consideration when training a model for the census dataset.

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task','classification'
          @census_model);
```

- The include_column_list option specifies feature columns that must be considered for training and should not be dropped.

  The following example specifies to consider the 'job' column when training a model for the census dataset.

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task','classification'
```

- The following example adds notes to the model_metadata.

```
mysql> CALL sys.ML_TRAIN('ml_data.iris_train', 'class', JSON_OBJECT('task', 'classification', 'notes', '
Query OK, 0 rows affected (1 min 42.53 sec)

mysql>  SELECT model_metadata FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@model;
+-----------------------------------------------------------+
| JSON_PRETTY(model_metadata)                               |
+-----------------------------------------------------------+
| {
  "task": "classification",
  "notes": "classification model",
  "chunks": 1,
  "format": "HWMLv2.0",
  "n_rows": 120,
  "status": "Ready",
  "options": {
```

```
    "task": "classification",
    "notes": "classification model",
    "model_explainer": "permutation_importance",
    "prediction_explainer": "permutation_importance"
  },
  "n_columns": 4,
  "column_names": [
    "sepal length",
    "sepal width",
    "petal length",
    "petal width"
  ],
  "contamination": null,
  "model_quality": "high",
  "training_time": 15.591492652893066,
  "algorithm_name": "SVC",
  "training_score": -0.03133905306458473,
  "build_timestamp": 1751897493,
  "hyperparameters": {
    "C": 47.004275502593885,
    "gamma": 0.000030517578125,
    "cache_size": 800,
    "class_weight": "balanced"
  },
  "n_selected_rows": 96,
  "training_params": {
    "recommend": "ratings",
    "force_use_X": false,
    "recommend_k": 3,
    "remove_seen": true,
    "ranking_topk": 10,
    "lsa_components": 100,
    "ranking_threshold": 1,
    "feedback_threshold": 1
  },
  "train_table_name": "ml_data.iris_train",
  "model_explanation": {
    "permutation_importance": {
      "petal width": 0.4194,
      "sepal width": 0.0,
      "petal length": 0.4192,
      "sepal length": 0.0415
    }
  },
  "n_selected_columns": 3,
  "target_column_name": "class",
  "optimization_metric": "neg_log_loss",
  "selected_column_names": [
    "petal length",
    "petal width",
    "sepal length"
  ],
  "training_drift_metric": {
    "mean": 0.0749,
    "variance": 0.0083
  }
} |
+-------------------------------------------------------------+
1 row in set (0.0416 sec)
```

See Also

- Train a Model
- The Model Catalog

7.1.2 ML_EXPLAIN

Running the ML_EXPLAIN routine on a model and dataset trains a prediction explainer and model explainer, and adds a model explanation to the model catalog. See Generate Model Explanations and Generate Prediction Explanations to learn more.

ML_EXPLAIN does not support recommendation, anomaly detection, and topic modeling models. A call with one of these models produces an error.

ML_EXPLAIN Syntax

```
mysql> CALL sys.ML_EXPLAIN ('table_name', 'target_column_name',
      model_handle, [options]);

options: {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['model_explainer', {'permutation_importance'|'partial_dependence'|'shap'|'fast_shap'}| NULL]
      ['prediction_explainer', {'permutation_importance'|'shap'}]
      ['columns_to_explain', JSON_ARRAY('column'[,'column'] ...)]
      ['target_value', 'target_class']
      }
}
```

When the ML_TRAIN routine runs, ML_EXPLAIN also runs with the Permutation Importance model explainer and prediction explainer. To run ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE with a different explainer, you must first run ML_EXPLAIN with the same explainer. See Generate Model Explanations and Generate Prediction Explanations to learn more.

Required ML_EXPLAIN Parameters

Set the following required parameters:

- table_name: You must define the table that you previously trained. The table name must be valid and fully qualified, so it must include the database name (*database_name.table_name*).
- target_column_name: Define the name of the target column in the training dataset that contains ground truth values.
- model_handle: Enter the model handle for the trained model. The model explanation is stored in this model metadata. The model must be loaded first. For example:

  ```
  mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', NULL);
  ```

  See Load a Model and Work with Model Handles to learn more.

  If you run ML_EXPLAIN again with the same model handle and model explainer, the model explanation field is overwritten with the new result.

ML_EXPLAIN Options

Optional parameters are specified as key-value pairs in JSON format. If an option is not specified, the default setting is used. If you specify NULL in place of the JSON argument, the default Permutation Importance model explainer is trained, and no prediction explainer is trained.

Set the following options as needed:

- model_explainer: Specifies one of the following model explainers:
  - permutation_importance: The default model explainer.

- `shap`: The SHAP model explainer, which produces feature importance values based on Shapley values.
- `fast_shap`: The Fast SHAP model explainer, which is a subsampling version of the SHAP model explainer. It usually has a faster runtime.
- `partial_dependence`: Explains how changing the values in one or more columns will change the value predicted by the model. The following additional arguments are required:
  - `columns_to_explain`: A JSON array of one or more column names in the table specified by *`table_name`*. The model explainer explains how changing the value in this column or columns affects the model.
  - `target_value`: A valid value that the target column containing ground truth values, as specified by `target_column_name`, can take.
- `prediction_explainer`: Specifies one of the following prediction explainers:
  - `permutation_importance`: The default prediction explainer.
  - `shap`: The SHAP prediction explainer, which produces feature importance values based on Shapley values.

Syntax Examples

Before running these examples, you must train and load the model first. See Train a Model and Load a Model.

- The following example sets `NULL` for the options, which trains the default Permutation Importance model explainer and no prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('bank_marketing_test.bank_train', 'y', @bank_test, NULL);
```

- The following example trains the Fast SHAP model explainer and SHAP prediction explainer.

```
mysql> CALL sys.ML_EXPLAIN('bank_marketing_test.bank_train', 'y', @bank_test, JSON_OBJECT('model_explainer',
```

- The following example trains the Partial Dependence model explainer (which requires extra options) and the SHAP prediction explainer. In this example, `sepal width` is the column to explain and the target value to include in `Iris_setosa`.

```
mysql> CALL sys.ML_EXPLAIN('ml_data.iris_train', 'class', @iris_model, JSON_OBJECT('columns_to_explain',
       JSON_ARRAY('sepal width'), 'target_value', 'Iris-setosa', 'model_explainer', 'partial_dependence', 'p
```

- You can query the model explanation from the model catalog. The `JSON_PRETTY` parameter displays the output in an easily readable format. See View Model Explanations.

```
mysql> SELECT JSON_PRETTY(model_explanation) FROM ML_SCHEMA_user1.MODEL_CATALOG WHERE model_handle=@census_m
+----------------------------------------------------------------------------------------------------------
| JSON_PRETTY(model_explanation)
+----------------------------------------------------------------------------------------------------------
| {
  "permutation_importance": {
    "age": 0.0292,
    "sex": 0.0023,
    "race": 0.0019,
    "fnlwgt": 0.0038,
    "education": 0.0008,
    "workclass": 0.0068,
```

```
    "occupation": 0.0223,
    "capital-gain": 0.0479,
    "capital-loss": 0.0117,
    "relationship": 0.0234,
    "education-num": 0.0352,
    "hours-per-week": 0.0148,
    "marital-status": 0.024,
    "native-country": 0.0
  }
} |
+-----------------------------------------------------------------------------------------------------------
1 row in set (0.0427 sec)
```

- An `ML_EXPLAIN` example that stores the model in the `model_object_catalog`.

```
mysql> SET @explain_option = JSON_OBJECT('model_explainer', 'shap', 'prediction_explainer', 'shap');
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.ML_EXPLAIN('mlcorpus.iris_train', 'class', @iris_model, @explain_option);
Query OK, 0 rows affected (11.51 sec)

mysql> SELECT model_object, model_object_size
      FROM ML_SCHEMA_user1.MODEL_CATALOG
      WHERE model_handle=@iris_model;
+--------------+-------------------+
| model_object | model_object_size |
+--------------+-------------------+
| NULL         |            348954 |
+--------------+-------------------+
1 row in set (0.00 sec)


mysql> SELECT model_metadata->>'$.format', model_metadata->>'$.chunks'
      FROM ML_SCHEMA_user1.MODEL_CATALOG
      WHERE model_handle=@iris_model;
+-----------------------------+-----------------------------+
| model_metadata->>'$.format' | model_metadata->>'$.chunks' |
+-----------------------------+-----------------------------+
| HWMLv2.0                    | 1                           |
+-----------------------------+-----------------------------+
1 row in set (0.00 sec)


mysql> SELECT chunk_id, length(model_object)
      FROM ML_SCHEMA_user1.model_object_catalog
      WHERE model_handle=@iris_model;
+----------+----------------------+
| chunk_id | length(model_object) |
+----------+----------------------+
|        1 |               348954 |
+----------+----------------------+
1 row in set (0.00 sec)
```

See Also

- Generate Model Explanations
- Generate Prediction Explanations

7.1.3 ML_MODEL_EXPORT

Use the `ML_MODEL_EXPORT` routine to export a model from the model catalog to a user defined table.

To learn how to use `ML_MODEL_EXPORT` to share models, see Grant Other Users Access to a Model.

ML_MODEL_EXPORT Overview

After you run ML_MODEL_EXPORT, the output table has these columns and formats:

- chunk_id:

  INT AUTO_INCREMENT PRIMARY KEY

- model_object:

  LONGTEXT DEFAULT NULL

- model_metadata:

  JSON

  See Model Metadata.

ML_MODEL_EXPORT should work regardless of model_metadata.status:

- If there is no corresponding row in the model_object_catalog for an existing model_handle in the MODEL_CATALOG:

  There should be only one row in the output table with chunk_id = 0, model_object = NULL and model_metadata = MODEL_CATALOG.model_metadata.

- If there is at least one row in the model_object_catalog for an existing model_handle in the MODEL_CATALOG:

  - There should be N rows in the output table with chunk_id being 1 to N.
  - ML_MODEL_EXPORT copies the model_object from model_object_catalog to the output table.
  - model_metadata in the row with chunk_id = 1 should be the same as in the MODEL_CATALOG.

ML_MODEL_EXPORT Syntax

```
mysql> CALL sys.ML_MODEL_EXPORT (model_handle, output_table_name);
```

ML_MODEL_EXPORT parameters:

- model_handle: The model handle for the model. See Work with Model Handles.
- output_table_name: The name for the output table.

Syntax Examples

- An example that exports a MySQL HeatWave AutoML model with metadata to the model catalog (ML_SCHEMA_user1.model_export). The output table name is model_export. You can then use SHOW_CREATE_TABLE to view information on the table for the exported model.

```
mysql> CALL sys.ML_MODEL_EXPORT(@iris_model, 'ML_SCHEMA_user1.model_export');
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW CREATE TABLE ML_SCHEMA_user1.model_export;
+--------------+------------------------------------------------------------------+
| Table        | Create Table                                                     |
+--------------+------------------------------------------------------------------+
| model_export | CREATE TABLE `model_export` (
`chunk_id` int NOT NULL AUTO_INCREMENT,
`model_object` longtext,
```

```
`model_metadata` json DEFAULT NULL,
PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+--------------------------------------------------------------------+
1 row in set (0.00 sec)
```

See Also

- Grant Other Users Access to a Model
- Manage External ONNX Models

7.1.4 ML_MODEL_IMPORT

Use the `ML_MODEL_IMPORT` routine to import a pre-trained model into your model catalog.

To learn how to use `ML_MODEL_IMPORT` to share models, see Grant Other Users Access to a Model.

ML_MODEL_IMPORT Overview

MySQL HeatWave AutoML supports the import of MySQL HeatWave AutoML and Open Neural Network Exchange (ONNX) format models. After import, all the MySQL HeatWave AutoML routines can be used with an ONNX model.

Models in ONNX format (`.onnx`) cannot be loaded directly into a MySQL table. They require string serialization and conversion to Base64 binary encoding. Before running `ML_MODEL_IMPORT`, follow the instructions in Import an External ONNX Model to pre-process and then load the model into a temporary table to use with AutoML.

The table to import should have the following columns, and their recommended parameters:

- `chunk_id`:

  `INT AUTO_INCREMENT PRIMARY KEY`
- `model_object`:

  `LONGTEXT NOT NULL`
- `model_metadata`:

  `JSON DEFAULT NULL`

  See Model Metadata.

The table must meet the following criteria:

- There must be only one row in the table with `chunk_id` = 1.
- The `model_metadata` corresponding to `chunk_id` = 1 must have the correct JSON key-value pair for the model format.

`ML_MODEL_IMPORT` stores the `model_metadata` corresponding to `chunk_id` = 1 in the model catalog, and ignores the `model_metadata` from other rows.

If `chunks` in the `model_metadata` corresponding to `chunk_id` = 1 is not set, it is set to the number of rows in the input table.

If `ML_MODEL_IMPORT` fails or is canceled, there is no change to the `MODEL_CATALOG` and to the `model_object_catalog`.

ML_MODEL_IMPORT Syntax

```
mysql> CALL sys.ML_MODEL_IMPORT (model_object, model_metadata, model_handle);

model_metadata (model from a table): {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['database', 'database']
      ['table', 'table']
      }
}

model_metadata (preprocessed model object): {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['task', {'classification'|'regression'|'forecasting'|'anomaly_detection'|'recommendation'}|NULL]
      ['build_timestamp', 'timestamp']
      ['target_column_name', 'column']
      ['train_table_name', 'table']
      ['column_names', JSON_ARRAY('column'[,'column'] ...)]
      ['model_explanation', ml_explain_options]
      ['notes', 'notes']
      ['format', 'format']
      ['status', {'creating'|'ready'|'error'}|NULL]
      ['model_quality', 'quality']
      ['training_time', 'time']
      ['algorithm_name', 'algorithm']
      ['training_score', 'score']
      ['n_rows', 'rows']
      ['n_columns', 'columns']
      ['n_selected_rows', 'rows']
      ['n_selected_columns', 'columns']
      ['optimization_metric', 'metric']
      ['selected_column_names', JSON_ARRAY('column'[,'column'] ...)]
      ['contamination', 'contamination']
      ['options', ml_train_options]
      ['training_params', ml_train_params]
      ['onnx_inputs_info', data_types_map]
      ['onnx_outputs_info', labels_map]
      ['training_drift_metric', JSON_OBJECT('mean', 'value', 'variance', 'value')]
      ['chunks', 'chunks']
}
```

ML_MODEL_IMPORT Parameters

Set the following parameters:

- `model_object`:
  - To import a model from a table: Set to `NULL`.
  - To import a model object: Define the preprocessed model object.
- `model_metadata`:
  - To import a model from a table:
    - `database`: The name of the database.
    - `table`: The name of the table.
  - To import a model object: An optional JSON object literal that contains key-value pairs with model metadata. See Model Metadata.

- `model_handle`: The model handle for the model. The model is stored in the model catalog under this name and accessed using it. Specify a model handle that does not already exist in the model catalog. Set to `NULL` for MySQL HeatWave AutoML to generate a unique model handle See Work with Model Handles.

Syntax Examples

- An example that imports a MySQL HeatWave AutoML model with metadata:

```
mysql> SET @hwml_model = "hwml_model";
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.ML_TRAIN('mlcorpus.input_train', 'target', NULL, @hwml_model);
Query OK, 0 rows affected (23.36 sec)

mysql> SET @hwml_object = (SELECT model_object FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=@hwm
Query OK, 0 rows affected (0.00 sec)

mysql> SET @hwml_metadata = (SELECT model_metadata FROM ML_SCHEMA_root.MODEL_CATALOG WHERE model_handle=
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.ML_MODEL_IMPORT(@hwml_object, @hwml_metadata, @imported_model);
Query OK, 0 rows affected (0.01 sec)

mysql> model_metadata->>'$.task' AS task, model_owner,
       model_metadata->>'$.build_timestamp' AS build_timestamp,
       model_metadata->>'$.target_column_name' AS target_column_name,
       model_metadata->>'$.train_table_name' AS train_table_name, model_object_size,
       model_metadata->>'$.model_explanation' AS model_explanation
       FROM ML_SCHEMA_root.MODEL_CATALOG;
+----------------+----------------+-----------------+--------------------+----------------------+-------
| task           | model_owner    | build_timestamp | target_column_name | train_table_name     | model_
+----------------+----------------+-----------------+--------------------+----------------------+-------
| classification | root           |      1679202888 | target             | mlcorpus.input_train |
| classification | root           |      1679202888 | target             | mlcorpus.input_train |
+----------------+----------------+-----------------+--------------------+----------------------+-------
2 rows in set (0.00 sec)


mysql> SELECT model_metadata->>'$.column_names' AS column_names FROM ML_SCHEMA_root.MODEL_CATALOG WHERE 
+-------------------------------------------------------------------------------------------+
| column_names                                                                              |
+-------------------------------------------------------------------------------------------+
| ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8", "C9", "C10", "C11", "C12", "C13"] |
| ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8", "C9", "C10", "C11", "C12", "C13"] |
+-------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

- An example that imports an ONNX model without specifying metadata:

```
mysql> CALL sys.ML_MODEL_IMPORT(@onnx_encode, NULL, 'onnx_test');
```

- An example that exports a model to a table, switches users, and then imports the model from that table:

```
mysql> CALL sys.ML_MODEL_EXPORT(@iris_model, 'ML_SCHEMA_user1.model_export');
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW CREATE TABLE ML_SCHEMA_user1.model_export;
+--------------+-------------------------------------------------------------------+
| Table        | Create Table                                                      |
+--------------+-------------------------------------------------------------------+
| model_export | CREATE TABLE `model_export` (
`chunk_id` int NOT NULL AUTO_INCREMENT,
`model_object` longtext,
`model_metadata` json DEFAULT NULL,
```

```
PRIMARY KEY (`chunk_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-------------------------------------------------------------------+
1 row in set (0.00 sec)

# switch to user2

mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'ML_SCHEMA_user1', 'table', 'model_export'), @ir
Query OK, 0 rows affected (0.19 sec)

mysql> CALL sys.ML_MODEL_LOAD(@iris_export, NULL);
Query OK, 0 rows affected (0.63 sec)

mysql> SELECT model_object, model_object_size FROM ML_SCHEMA_user2.MODEL_CATALOG WHERE model_handle=@iris_ex
+--------------+-------------------+
| model_object | model_object_size |
+--------------+-------------------+
| NULL         |            348954 |
+--------------+-------------------+
1 row in set (0.00 sec)


mysql> SELECT chunk_id, LENGTH(model_object) FROM ML_SCHEMA_user2.model_object_catalog WHERE model_handle=@i
+----------+----------------------+
| chunk_id | LENGTH(model_object) |
+----------+----------------------+
|        1 |               348954 |
+----------+----------------------+
1 row in set (0.00 sec)
```

- An example that imports a model in ONNX format from a table:

```
mysql> DROP TABLE IF EXISTS model_table;

mysql> CREATE TABLE model_table (chunk_id INT AUTO_INCREMENT PRIMARY KEY, model_object LONGTEXT NOT NULL, mo

mysql> LOAD DATA INFILE '/onnx_examples/x00'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (34.96 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x01'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (32.74 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE '/onnx_examples/x02'
       INTO TABLE model_table
       CHARACTER SET binary
       FIELDS TERMINATED BY '\t'
       LINES TERMINATED BY '\r'
       (model_object);
Query OK, 1 row affected (11.90 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SET @model_metadata = JSON_OBJECT('task','classification','onnx_outputs_info',JSON_OBJECT('prediction

mysql> UPDATE mlcorpus.model_table SET model_metadata=@model_metadata WHERE chunk_id=1;
```

```
mysql> CALL sys.ML_MODEL_IMPORT(NULL, JSON_OBJECT('schema', 'mlcorpus', 'table', 'model_table'), @onnx_mo
Query OK, 0 rows affected (18 min 7.29 sec)

mysql> CALL sys.ML_MODEL_LOAD(@onnx_model, NULL);
Query OK, 0 rows affected (6 min 51.37 sec)

mysql> SELECT COUNT(*) FROM ML_SCHEMA_root.model_object_catalog WHERE model_handle=@onnx_model;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT SUM(LENGTH(model_object)) FROM ML_SCHEMA_root.model_object_catalog WHERE model_handle=@onnx
+---------------------------+
| SUM(LENGTH(model_object)) |
+---------------------------+
|                2148494845 |
+---------------------------+
1 row in set (57.36 sec)
```

7.1.5 ML_PREDICT_ROW

ML_PREDICT_ROW generates predictions for one or more rows of unlabeled data specified in JSON format. Invoke ML_PREDICT_ROW with a SELECT statement.

A call to ML_PREDICT_ROW can include columns that were not present during ML_TRAIN. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and predictions in the same table. ML_PREDICT_ROW ignores any extra columns, and appends them to the results.

ML_PREDICT_ROW Syntax

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle), [options]);

options: {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['threshold', 'N']
      ['topk', 'N']
      ['recommend', {'ratings'|'items'|'users'|'users_to_items'|'items_to_users'|'items_to_items'|'users_to
      ['remove_seen', {'true'|'false'}]
      ['additional_details', {'true'|'false'}]
      }
}
```

Required ML_PREDICT_ROW Parameters

Set the following required parameters:

- input_data: Define the data to generate predictions for. The column names must match the feature column names in the table used to train the model. You can define the input data in the following ways:

  Specify a single row of data in JSON format.

  ```
  mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("column_name", value, "column_name", value, ...), model_hand
  ```

  Run ML_PREDICT_ROW on multiple rows of data by specifying the columns as key-value pairs in JSON format and select from a table.

  ```
  mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, "output_col_name
          FROM input_table_name LIMIT N;
  ```

- `model_handle`: Define the model handle or a session variable that contains the model handle. See Work with Model Handles.

Review the following options in `JSON` format.

ML_PREDICT ROW Option for Data Drift Detection

To view data drift detection values for classification and regression models, set the `additional_details` option to `true`. The `ml_results` includes the `drift` JSON object literal. See Analyze Data Drift.

ML_PREDICT_ROW Options for Recommendation Models

Set the following options as needed for Recommendation models.

- `topk`: Specify the number of recommendations to provide as a positive integer. The default is `3`.
- `recommend`: Specify what to recommend.
  - `ratings`: Use this option to predict ratings. This is the default value.

    The target column is `prediction`, and the values are `float`.

    The input table must contain at least two columns with the same names as the user column and item column from the training model.
  - `items`: Use this option to recommend items for users.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"), "column_rating_name" , JSON_ARRAY
    ```

    The input table must contain at least one column with the same name as the user column from the training model.
  - `users`: Use this option to recommend users for items.

    The target column is `user_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"), "column_rating_name" , JSON_ARRAY
    ```

    The input table must contain at least one column with the same name as the item column from the training model.
  - `users_to_items`: This is the same as `items`.
  - `items_to_users`: This is the same as `users`.
  - `items_to_items`: Use this option to recommend similar items for items.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"))
    ```

    The input table must contain at least contain a column with the same name as the item column from the training model.
  - `users_to_users`: Use this option to recommend similar users for users.

The target column is `user_recommendation`, and the values are:

```
JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"))
```

The input table must contain at least one column with the same name as the user column from the training model.

- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Options for Anomaly Detection Models

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.

Syntax Examples

- The following example generates a prediction on a single row of data. The results include the `ml_results` field, which uses `JSON` format. Optionally use `\G` to display the information in an easily readable format.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 25,
          "workclass", "Private",
          "fnlwgt", 226802,
          "education", "11th",
          "education-num", 7,
          "marital-status", "Never-married",
          "occupation", "Machine-op-inspct",
          "relationship", "Own-child",
          "race", "Black",
          "sex", "Male",
          "capital-gain", 0,
          "capital-loss", 0,
          "hours-per-week", 40,
          "native-country", "United-States");
mysql> SELECT sys.ML_PREDICT_ROW(@row_input, @census_model, NULL)\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(@row_input, @census_model, NULL):
{
    "age": 25,
    "sex": "Male",
    "race": "Black",
    "fnlwgt": 226802,
    "education": "11th",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "probabilities": {
            ">50K": 0.0032,
            "<=50K": 0.9968
        }
    },
```

```
    "occupation": "Machine-op-inspct",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Own-child",
    "education-num": 7,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States"
}
1 row in set (2.2218 sec)
```

- The following example generates predictions on two rows of data from the input table. Optionally use `\G` to display the information in an easily readable format.

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, NULL)FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                            "age": 62,
                                            "sex": "Female",
                                            "race": "White",
                                            "fnlwgt": 123582,
                                            "education": "10th",
                                            "workclass": "Private",
                                            "Prediction": "<=50K",
                                            "ml_results": {
                                                "predictions": {
                                                    "revenue": "<=50K"
                                                },
                                                "probabilities": {
                                                    ">50K": 0.0106,
                                                    "<=50K": 0.9894
                                                }
                                            },
                                            "occupation": "Other-service",
                                            "capital-gain": 0,
                                            "capital-loss": 0,
                                            "relationship": "Unmarried",
                                            "education-num": 6,
                                            "hours-per-week": 40,
                                            "marital-status": "Divorced",
                                            "native-country": "United-States"
                                        }
*************************** 2. row ***************************
sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_train.`age`,
"workclass", census_train.`workclass`,
```

```
"fnlwgt", census_train.`fnlwgt`,
"education", census_train.`education`,
"education-num", census_train.`education-num`,
"marital-status", census_train.`marita: {
                                            "age": 32,
                                            "sex": "Female",
                                            "race": "White",
                                            "fnlwgt": 174215,
                                            "education": "Bachelors",
                                            "workclass": "Federal-gov",
                                            "Prediction": "<=50K",
                                            "ml_results": {
                                                "predictions": {
                                                    "revenue": "<=50K"
                                                },
                                                "probabilities": {
                                                    ">50K": 0.3249,
                                                    "<=50K": 0.6751
                                                }
                                            },
                                            "occupation": "Exec-managerial",
                                            "capital-gain": 0,
                                            "capital-loss": 0,
                                            "relationship": "Not-in-family",
                                            "education-num": 13,
                                            "hours-per-week": 60,
                                            "marital-status": "Never-married",
                                            "native-country": "United-States"
                                        }
2 rows in set (9.6548 sec)
```

- The following example uses explicit feedback and runs the `ML_PREDICT_ROW` routine to predict the top 3 items that a particular user will like with the `users_to_items` option.

```
mysql> SELECT sys.ML_PREDICT_ROW('{"user_id": "846"}', @model, JSON_OBJECT("recommend", "users_to_items"
+-------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW('{"user_id": "846"}', @model,  JSON_OBJECT("recommend", "users_to_items", "topk", 3
+-------------------------------------------------------------------------------------------------------
| {"user_id": "846", "ml_results": "{"predictions": {"item_id": ["313", "483", "64"], "rating": [4.06, 4
+-------------------------------------------------------------------------------------------------------
1 row in set (0.2811 sec)
```

- The following example generates predictions on ten rows from an input table. The `additional_details` parameter is set to `TRUE, so you can review data drift details.`

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT(
 "age", census_test.`age`,
 "workclass", census_test.`workclass`,
 "fnlwgt", census_test.`fnlwgt`,
 "education", census_test.`education`,
 "education-num", census_test.`education-num`,
 "marital-status", census_test.`marital-status`,
 "occupation", census_test.`occupation`,
 "relationship", census_test.`relationship`,
 "race", census_test.`race`,
 "sex", census_test.`sex`,
 "capital-gain", census_test.`capital-gain`,
 "capital-loss", census_test.`capital-loss`,
 "hours-per-week", census_test.`hours-per-week`,
 "native-country", census_test.`native-country`),
 @census_model, JSON_OBJECT('additional_details', TRUE))FROM census_data.census_test LIMIT 10;
+-------------------------------------------------------------------------------------------------------
| sys.ML_PREDICT_ROW(JSON_OBJECT(
"age", census_test.`age`,
"workclass", census_test.`workclass`,
"fnlwgt", census_test.`fnlwgt`,
```

```
"education", census_test.`education`,
"education-num", census_test.`education-num`,
"ma
+------------------------------------------------------------------------------------------------------
| {"age": 37, "sex": "Male", "race": "White", "fnlwgt": 99146, "education": "Bachelors", "workclass": "Priva
| {"age": 34, "sex": "Male", "race": "White", "fnlwgt": 27409, "education": "9th", "workclass": "Private", "
| {"age": 30, "sex": "Female", "race": "White", "fnlwgt": 299507, "education": "Assoc-acdm", "workclass": "P
| {"age": 62, "sex": "Female", "race": "White", "fnlwgt": 102631, "education": "Some-college", "workclass":
| {"age": 51, "sex": "Male", "race": "White", "fnlwgt": 153486, "education": "Some-college", "workclass": "P
| {"age": 34, "sex": "Male", "race": "Black", "fnlwgt": 434292, "education": "HS-grad", "workclass": "Privat
| {"age": 28, "sex": "Male", "race": "White", "fnlwgt": 240172, "education": "Masters", "workclass": "Self-e
| {"age": 56, "sex": "Male", "race": "White", "fnlwgt": 219426, "education": "10th", "workclass": "Private",
| {"age": 46, "sex": "Female", "race": "White", "fnlwgt": 295791, "education": "HS-grad", "workclass": "Priv
| {"age": 46, "sex": "Male", "race": "White", "fnlwgt": 114032, "education": "Some-college", "workclass": "P
+------------------------------------------------------------------------------------------------------
10 rows in set (6.8109 sec)
```

See Also

- Generate Predictions for a Row of Data
- Analyze Data Drift

7.1.6 ML_PREDICT_TABLE

`ML_PREDICT_TABLE` generates predictions for an entire table of unlabeled data. MySQL HeatWave AutoML performs the predictions in parallel.

This topic has the following sections.

- ML_PREDICT_TABLE Overview
- ML_PREDICT_TABLE Syntax
- Required ML_PREDICT_TABLE Parameters
- ML_PREDICT_TABLE Options
- Options for Recommendation Models
- Requirements and Options for Anomaly Detection Models
- Options for Forecasting Models
- Syntax Examples
- See Also

ML_PREDICT_TABLE Overview

`ML_PREDICT_TABLE` is a compute intensive process. If `ML_PREDICT_TABLE` takes a long time to complete, manually limit input tables to a maximum of 1,000 rows.

A call to `ML_PREDICT_TABLE` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and predictions in the same table. `ML_PREDICT_TABLE` ignores any extra columns, and appends them to the results.

The output table includes a primary key:

- If the input table has a primary key, the output table has the same primary key.

- If the input table does not have a primary key, the output table has a new primary key column that auto increments. The name of the new primary key column is `_4aad19ca6e_pk_id`. The input table must not have a column with the name `_4aad19ca6e_pk_id` that is not a primary key.

The output of predictions includes the `ml_results` column, which contains the prediction results and the data. The combination of results and data must be less than 65,532 characters.

You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

`ML_PREDICT_TABLE` supports data drift detection for classification and regression models with the following:

- The *`options`* parameter includes the `additional_details` boolean value.
- The `ml_results` column includes the `drift` JSON object literal.

See Analyze Data Drift.

ML_PREDICT_TABLE Syntax

```
mysql> CALL sys.ML_PREDICT_TABLE(table_name, model_handle, output_table_name), [options]);

options: {
     JSON_OBJECT("key","value"[,"key","value"] ...)
          "key","value": {
          ['threshold', 'N']
          ['topk', 'N']
          ['recommend', {'ratings'|'items'|'users'|'users_to_items'|'items_to_users'|'items_to_items'|'use
          ['remove_seen', {'true'|'false'}]
                    ['additional_details', {'true'|'false'}]
          ['prediction_interval', 'N']
          ['logad_options', JSON_OBJECT(("key","value"[,"key","value"] ...)
                 "key","value": {
                              ['summarize_logs', {'true'|'false'}]
                              ['summary_threshold', 'N']
                              }
          }
}
```

Required ML_PREDICT_TABLE Parameters

Set the following required parameters:

- `table_name`: Specifies the fully qualified name of the input table (*`database_name.table_name`*). The input table should contain the same feature columns as the training dataset. If the target column is included in the input table, it is not considered when generating predictions.
- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- `output_table_name`: Specifies the table where predictions are stored. A fully qualified table name must be specified (*`database_name.table_name`*). You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

ML_PREDICT_TABLE Options

Set the following options in `JSON` format as needed.

- To view data drift detection values for classification and regression models, set the `additional_details` option to `true`. The `ml_results` includes the `drift` JSON object literal.

Additional options are available for recommendation, anomaly detection, and forecasting models.

Options for Recommendation Models

Set the following options as needed for recommendation models.

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The optional top number of recommendations to provide. The default is `3`. Set a positive integer between 1 and the number of rows in the table.

  A `recommendation` task with implicit feedback can use both `threshold` and `topk`.
- `recommend`: Specify what to recommend.
  - `ratings`: Use this option to predict ratings. This is the default value.

    The target column is `prediction`, and the values are `float`.

    The input table must contain at least two columns with the same names as the user column and item column from the training model.
  - `items`: Use this option to recommend items for users.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"), "column_rating_name" , JSON_ARRAY
    ```

    The input table must contain at least one column with the same name as the user column from the training model.
  - `users`: Use this option to recommend users for items.

    The target column is `user_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"), "column_rating_name" , JSON_ARRAY
    ```

    The input table must contain at least one column with the same name as the item column from the training model.
  - `users_to_items`: This is the same as `items`.
  - `items_to_users`: This is the same as `users`.
  - `items_to_items`: Use this option to recommend similar items for items.

    The target column is `item_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_item_id_name", JSON_ARRAY("item_1", ... , "item_k"))
    ```

    The input table must contain at least one column with the same name as the item column from the training model.
  - `users_to_users`: Use this option to recommend similar users for users.

    The target column is `user_recommendation`, and the values are:

    ```
    JSON_OBJECT("column_user_id_name", JSON_ARRAY("user_1", ... , "user_k"))
    ```

The input table must at least contain a column with the same name as the user column from the training model.

- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Requirements and Options for Anomaly Detection Models

If you run `ML_PREDICT_TABLE` with the `log_anomaly_detection` task, at least one column must act as the primary key to establish the temporal order of logs.

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.

- `topk`: The optional top K rows to display with the highest anomaly scores. Set a positive integer between 1 and the number of rows in the table. If `topk` is not set, `ML_PREDICT_TABLE` uses `threshold`.

  Do not set both `threshold` and `topk`. Use `threshold` or `topk`, or set *`options`* to `NULL`.

- `logad_options`: A `JSON_OBJECT` that allows you to configure the following options for running an anomaly detection model on log data.

  - `summarize_logs`: Allows you to leverage MySQL HeatWave GenAI to generate textual summaries of results. Enable this option by setting it to `TRUE`. If enabled, summaries are generated for log segments that are labeled as an anomaly or have anomaly scores higher than the value set for the `summary_threshold`.

  - `summary_threshold`: Determines the rows in the output table that are summarized. This does not affect how the `contamination` and `threshold` options determine anomalies. You can set a value greater than 0 and less than 1. The default value is `NULL`. If `NULL` is selected, only the log segments tagged with `is_anomaly` are used to generate summaries.

Options for Forecasting Models

Set the following options as needed for forecasting models.

- `prediction_interval`: Use this to generate forecasted values with lower and upper bounds based on a specific prediction interval (level of confidence). For the `prediction_interval` value:

  - The default value is 0.95.

  - The data type for this value must be FLOAT.

  - The value must be greater than 0 and less than 1.

Syntax Examples

- A typical usage example that specifies the fully qualified name of the table to generate predictions for, the session variable containing the model handle, and the fully qualified output table name.

```
mysql> CALL sys.ML_PREDICT_TABLE('census_data.census_train', @census_model, 'census_data.census_train_predic
```

To view `ML_PREDICT_TABLE` results, query the output table. The table shows the predictions and the feature column values used to make each prediction. The table includes the primary key, `_4aad19ca6e_pk_id`, and the `ml_results` column, which uses `JSON` format:

```
mysql> SELECT * FROM census_train_predictions LIMIT 5;
+-------------------+-----+------------------+--------+--------------+--------------+-------------------+-
| _4aad19ca6e_pk_id | age | workclass        | fnlwgt | education    | education-num | marital-status     |
+-------------------+-----+------------------+--------+--------------+--------------+-------------------+-
|                 1 |  37 | Private          |  99146 | Bachelors    |            13 | Married-civ-spouse |
|                 2 |  34 | Private          |  27409 | 9th          |             5 | Married-civ-spouse |
|                 3 |  30 | Private          | 299507 | Assoc-acdm   |            12 | Separated          |
|                 4 |  62 | Self-emp-not-inc | 102631 | Some-college |            10 | Widowed            |
|                 5 |  51 | Private          | 153486 | Some-college |            10 | Married-civ-spouse |
+-------------------+-----+------------------+--------+--------------+--------------+-------------------+-
5 rows in set (0.0014 sec)
```

- The following example generates a table of recommendations. The output recommends the top three items that particular users will like.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.test_sample',  @model, 'mlcorpus.table_predictions_users',  JSON_
Query OK, 0 rows affected (5.0672 sec)

mysql> SELECT * FROM mlcorpus.table_predictions_users LIMIT 3;
+-------------------+---------+---------+--------+--------------------------------------------------------------
| _4aad19ca6e_pk_id | user_id | item_id | rating | ml_results
+-------------------+---------+---------+--------+--------------------------------------------------------------
|                 1 | 1026    | 13763   |      1 | {"predictions": {"item_id": ["10", "14", "11"], "rating":
|                 2 | 992     | 16114   |      1 | {"predictions": {"item_id": ["10", "14", "11"], "rating":
|                 3 | 1863    | 4527    |      1 | {"predictions": {"item_id": ["10", "14", "11"], "rating":
+-------------------+---------+---------+--------+--------------------------------------------------------------
```

- The following example generates a table of anomaly detection predictions. A threshold value of 1% is specified.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', @anomaly, 'mlcorpu
Query OK, 0 rows affected (12.77 sec)

mysql> SELECT * FROM mlcorpus_anomaly_detection.volcanoes-predictions_threshold LIMIT 5;
+-------------------+------+------+----------+--------+-------------------------------------------------------
| _4aad19ca6e_pk_id | V1   | V2   | V3       | target | ml_results
+-------------------+------+------+----------+--------+-------------------------------------------------------
|                 1 |  128 |  802 |  0.47255 |      0 | {'predictions': {'is_anomaly': 1}, 'probabilities':
|                 2 |  631 |  642 | 0.387302 |      0 | {'predictions': {'is_anomaly': 1}, 'probabilities':
|                 3 |  438 |  959 | 0.556034 |      0 | {'predictions': {'is_anomaly': 1}, 'probabilities':
|                 4 |  473 |  779 | 0.407626 |      0 | {'predictions': {'is_anomaly': 1}, 'probabilities':
|                 5 |   67 |  933 | 0.383843 |      0 | {'predictions': {'is_anomaly': 1}, 'probabilities':
+-------------------+------+------+----------+--------+-------------------------------------------------------
5 rows in set (0.00 sec)
```

- The following example generates a table of anomaly detection predictions by using semi-supervised learning. It overrides the `ensemble_score` value from the `ML_TRAIN` routine to a new value of 0.5.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.anomaly_train',@semsup_gknn, 'mlcorpus.preds_gknn_weighted', CAST
```

- The following example generates a table of anomaly detection predictions for log data. It disables log summaries in the results.

```
mysql> CALL sys.ML_PREDICT_TABLE('mlcorpus.`log_anomaly_just_patterns`', @logad_model, 'mlcorpus.log_anomaly
mysql> SELECT * FROM mlcorpus.log_anomaly_test_out LIMIT 1;
+----+--------------------------------------------------------------------------------------------------------
| id | parsed_log_segment
+----+--------------------------------------------------------------------------------------------------------
```

```
|  1 | 2024-04-11T14:39:45.443597Z 1 [Note] [MY-013546] [InnoDB] Atomic write enabled
|    | 2024-04-11T14:39:45.443618Z 1 [Note] [MY-012932] [InnoDB] PUNCH HOLE support available
|    | 2024-04-11T14:39:45.443631Z 1 [Note] [MY-012944] [InnoDB] Uses event mutexes
|    | 2024-04-11T14:39:45.443635Z 1 [Note] [MY-012945] [InnoDB] GCC builtin __atomic_thread_fence() is
|    | 2024-04-11T14:39:45.443646Z 1 [Note] [MY-012948] [InnoDB] Compressed tables use zlib 1.2.13
|    | 2024-04-11T14:40:25.128143Z 0 [Note] [MY-010264] [Server] - '127.0.0.1' resolves to '127.0.0.1';
|    | 2024-04-11T14:40:25.128182Z 0 [Note] [MY-010251] [Server] Server socket created on IP: '127.0.0.1
|    | 2024-04-11T14:40:25.128245Z 0 [Note] [MY-010252] [Server] Server hostname (bind-address): '10.0.1
|    | 2024-04-11T14:40:25.128272Z 0 [Note] [MY-010264] [Server] - '10.0.1.125' resolves to '10.0.1.125'
|    | 2024-04-26T13:01:30.287325Z 0 [Warning] [MY-015116] [Server] Background histogram update on nexus
|    | Lock wait timeout exceeded; try restarting transaction
+----+------+------+----------+--------+----------------------------------------------------------------
```

See Also

- Generate Predictions for a Table
- Analyze Data Drift

7.1.7 ML_EXPLAIN_ROW

The `ML_EXPLAIN_ROW` routine generates explanations for one or more rows of unlabeled data. Invoke `ML_EXPLAIN_ROW` with a `SELECT` statement. It limits explanations to the 100 most relevant features.

A loaded and trained model with the appropriate prediction explainer is required to run `ML_EXPLAIN_ROW`. See Generate Prediction Explanations for a Row of Data.

`ML_EXPLAIN_ROW` does not support recommendation, anomaly detection and topic modeling models. A call with one of these models produces an error.

A call to `ML_EXPLAIN_ROW` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and explanations in the same table. `ML_EXPLAIN_ROW` ignores any extra columns, and appends them to the results.

ML_EXPLAIN_ROW Syntax

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);

options: {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['prediction_explainer', {'permutation_importance'|'shap'}|NULL]
      }
}
```

Required ML_EXPLAIN_ROW Parameters

Set the following required parameters:

- `input_data`: Define the data to generate explanations for. The column names must match the feature column names in the table used to train the model. You can define the input data in the following ways:

  Specify a single row of data in `JSON` format:

  ```
  mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("column_name", value, "column_name", value, ...)', model_ha
  ```

  To run `ML_EXPLAIN_ROW` on multiple rows of data, specify the columns in `JSON` key-value format and select from an input table:

  ```
  mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("output_col_name", schema.`input_col_name`, output_col_name
  ```

```
        FROM input_table_name LIMIT N;
```

- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.

ML_EXPLAIN_ROW Options

You can set the following option in JSON format as needed:

- `prediction_explainer`: The name of the prediction explainer that you have trained for this model using `ML_EXPLAIN`.
  - `permutation_importance`: The default prediction explainer.
  - `shap`: The SHAP prediction explainer, which produces feature importance values based on Shapley values.

Syntax Examples

- The following example generates a prediction explainer on a single row of data with the default Permutation Importance prediction explainer. The results include the `ml_results` field, which uses `JSON` format. Optionally, use `\G` to display the output in an easily readable format.

```
mysql> SET @row_input = JSON_OBJECT(
          "age", 31,
          "workclass", "Private",
          "fnlwgt", 45781,
          "education", "Masters",
          "education-num", 14,
          "marital-status", "Married-civ-spouse",
          "occupation", "Prof-specialty",
          "relationship", "Not-in-family",
          "race", "White",
          "sex", "Female",
          "capital-gain", 14084,
          "capital-loss", 2042,
          "hours-per-week", 40,
          "native-country", "India");
mysql> SELECT sys.ML_EXPLAIN_ROW(@row_input, @census_model, JSON_OBJECT('prediction_explainer', 'permutation
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(@row_input, @census_model,
          JSON_OBJECT('prediction_explainer', 'permutation_importance')):
          {
                "age": 31,
                "sex": "Female",
                "race": "White",
                "Notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                "fnlwgt": 45781,
                "education": "Masters",
                "workclass": "Private",
                "Prediction": ">50K",
                "ml_results": {
                   "notes": "capital-gain (14084) had the largest impact towards predicting >50K",
                   "predictions": {
                       "revenue": ">50K"
                    },
                    "attributions": {
                        "age": 0.34,
                        "sex": 0,
                        "race": 0,
                        "fnlwgt": 0,
                        "education": 0,
                        "workclass": 0,
                        "occupation": 0,
```

```
                        "capital-gain": 0.97,
                        "capital-loss": 0,
                        "relationship": 0,
                        "education-num": 0.04,
                        "hours-per-week": 0,
                        "marital-status": 0
                    }
                },
                "occupation": "Prof-specialty",
                "capital-gain": 14084,
                "capital-loss": 2042,
                "relationship": "Not-in-family",
                "education-num": 14,
                "hours-per-week": 40,
                "marital-status": "Married-civ-spouse",
                "native-country": "India",
                "age_attribution": 0.34,
                "sex_attribution": 0,
                "race_attribution": 0,
                "fnlwgt_attribution": 0,
                "education_attribution": 0,
                "workclass_attribution": 0,
                "occupation_attribution": 0,
                "capital-gain_attribution": 0.97,
                "capital-loss_attribution": 0,
                "relationship_attribution": 0,
                "education-num_attribution": 0.04,
                "hours-per-week_attribution": 0,
                "marital-status_attribution": 0
          }
1 row in set (6.3072 sec)
```

- The following example generates prediction explainers on two rows of the input table with the SHAP prediction explainer.

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT(
 "age", census_train.`age`,
 "workclass", census_train.`workclass`,
 "fnlwgt", census_train.`fnlwgt`,
 "education", census_train.`education`,
 "education-num", census_train.`education-num`,
 "marital-status", census_train.`marital-status`,
 "occupation", census_train.`occupation`,
 "relationship", census_train.`relationship`,
 "race", census_train.`race`,
 "sex", census_train.`sex`,
 "capital-gain", census_train.`capital-gain`,
 "capital-loss", census_train.`capital-loss`,
 "hours-per-week", census_train.`hours-per-week`,
 "native-country", census_train.`native-country`),
 @census_model, JSON_OBJECT('prediction_explainer', 'shap'))FROM census_data.census_train LIMIT 2\G
*************************** 1. row ***************************
sys.ML_EXPLAIN_ROW(JSON_OBJECT( "age", census_train.`age`, "workclass", census_train.`workclass`, "fnlwg
    "age": 22,
    "sex": "Female",
    "race": "Black",
    "fnlwgt": 310380,
    "education": "HS-grad",
    "workclass": "Private",
    "Prediction": "<=50K",
    "ml_results": {
        "predictions": {
            "revenue": "<=50K"
        },
        "attributions": {
            "age_attribution": 0.055990096751945995,
            "sex_attribution": 0.011676016319165776,
```

```
            "race_attribution": 0.005258734090653583,
            "fnlwgt_attribution": 0,
            "education_attribution": 0,
            "workclass_attribution": 0,
            "occupation_attribution": 0.0036531218497025536,
            "capital-gain_attribution": 0.017052572967215754,
            "capital-loss_attribution": 0,
            "relationship_attribution": 0.03019321048408115,
            "education-num_attribution": 0.01749651048882997,
            "hours-per-week_attribution": 0.003671861337781857,
            "marital-status_attribution": 0.03869036669327783
        }
    },
    "occupation": "Adm-clerical",
    "capital-gain": 0,
    "capital-loss": 0,
    "relationship": "Unmarried",
    "education-num": 9,
    "hours-per-week": 40,
    "marital-status": "Never-married",
    "native-country": "United-States",
    "age_attribution": 0.0559900968,
    "sex_attribution": 0.0116760163,
    "race_attribution": 0.0052587341,
    "fnlwgt_attribution": 0,
    "education_attribution": 0,
    "workclass_attribution": 0,
    "occupation_attribution": 0.0036531218,
    "capital-gain_attribution": 0.017052573,
    "capital-loss_attribution": 0,
    "relationship_attribution": 0.0301932105,
    "education-num_attribution": 0.0174965105,
    "hours-per-week_attribution": 0.0036718613,
    "marital-status_attribution": 0.0386903667
}
*************************** 2. row ***************************
sys.ML_EXPLAIN_ROW(JSON_OBJECT( "age", census_train.`age`, "workclass", census_train.`workclass`, "fnlwgt",
    "age": 45,
    "sex": "Male",
    "race": "White",
    "fnlwgt": 182100,
    "education": "Bachelors",
    "workclass": "Local-gov",
    "Prediction": ">50K",
    "ml_results": {
        "predictions": {
            "revenue": ">50K"
        },
        "attributions": {
            "age_attribution": 0.10591945090998228,
            "sex_attribution": 0.013172526260700925,
            "race_attribution": 0.007606345008707882,
            "fnlwgt_attribution": 0.018097167152459265,
            "education_attribution": -0.007944704365873384,
            "workclass_attribution": 0.01615429281764716,
            "occupation_attribution": 0.08573874801531925,
            "capital-gain_attribution": -0.003364275424074914,
            "capital-loss_attribution": 0,
            "relationship_attribution": 0.099373669980131,
            "education-num_attribution": 0.1380689603088001,
            "hours-per-week_attribution": 0.0124334565747376,
            "marital-status_attribution": 0.0938256104928338
        }
    },
    "occupation": "Sales",
    "capital-gain": 0,
    "capital-loss": 0,
```

```
    "relationship": "Husband",
    "education-num": 13,
    "hours-per-week": 40,
    "marital-status": "Married-civ-spouse",
    "native-country": "United-States",
    "age_attribution": 0.1059194509,
    "sex_attribution": 0.0131725263,
    "race_attribution": 0.007606345,
    "fnlwgt_attribution": 0.0180971672,
    "education_attribution": -0.0079447044,
    "workclass_attribution": 0.0161542928,
    "occupation_attribution": 0.085738748,
    "capital-gain_attribution": -0.0033642754,
    "capital-loss_attribution": 0,
    "relationship_attribution": 0.09937367,
    "education-num_attribution": 0.1380689603,
    "hours-per-week_attribution": 0.0124334566,
    "marital-status_attribution": 0.0938256105
}
2 rows in set (5.5382 sec)
```

See Also

- Generate Prediction Explanations for a Row of Data

7.1.8 ML_EXPLAIN_TABLE

`ML_EXPLAIN_TABLE` explains predictions for an entire table of unlabeled data. It limits explanations to the 100 most relevant features.

ML_EXPLAIN_TABLE Overview

Note

`ML_EXPLAIN_TABLE` is a very memory-intensive process. We recommend limiting the input table to a maximum of 100 rows. If the input table has more than ten columns, limit it to ten rows.

A call to `ML_EXPLAIN_TABLE` can include columns that were not present during `ML_TRAIN`. A table can include extra columns, and still use the MySQL HeatWave AutoML model. This allows side by side comparisons of target column labels, ground truth, and explanations in the same table. `ML_EXPLAIN_TABLE` ignores any extra columns, and appends them to the results.

A loaded model and trained with the appropriate prediction explainer is required to run `ML_EXPLAIN_TABLE`. See Generate Prediction Explanations for a Table.

The output table includes a primary key:

- If the input table has a primary key, the output table will have the same primary key.
- If the input table does not have a primary key, the output table will have a new primary key column that auto increments. The name of the new primary key column is `_4aad19ca6e_pk_id`. The input table must not have a column with the name `_4aad19ca6e_pk_id` that is not a primary key.

You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

`ML_EXPLAIN_TABLE` does not support recommendation, anomaly detection, and topic modeling models. A call with one of these models produces an error.

ML_EXPLAIN_TABLE Syntax

```
mysql> CALL sys.ML_EXPLAIN_TABLE(table_name, model_handle, output_table_name, [options]);

options: {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['prediction_explainer', {'permutation_importance'|'shap'}|NULL]
            }
}
```

Required ML_EXPLAIN_TABLE Parameters

Set the following required parameters.

- `table_name`: Specifies the fully qualified name of the input table (*`database_name.table_name`*). The input table should contain the same feature columns as the table used to train the model. If the target column is included in the input table, it is not considered when generating prediction explanations.
- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- `output_table_name`: Specifies the table where explanation data is stored. A fully qualified table name must be specified (*`database_name.table_name`*). You have the option to specify the input table and output table as the same table if specific conditions are met. See Input Tables and Output Tables to learn more.

ML_EXPLAIN_TABLE Options

Set the following options as needed.

- `prediction_explainer`: The name of the prediction explainer that you have trained for this model using `ML_EXPLAIN`.
  - `permutation_importance`: The default prediction explainer.
  - `shap`: The SHAP prediction explainer, which produces feature importance values based on Shapley values.

Syntax Examples

- The following example generates explanations for a table of data with the default Permutation Importance prediction explainer. The `ML_EXPLAIN_TABLE` call specifies the fully qualified name of the table to generate explanations for, the session variable containing the model handle, and the fully qualified output table name.

  ```
  mysql> CALL sys.ML_EXPLAIN_TABLE('census_data.census_train', @census_model, 'census_data.census_train_permut
  ```

  To view `ML_EXPLAIN_TABLE` results, query the output table. The `SELECT` statement retrieves explanation data from the output table. The table includes the primary key, `_4aad19ca6e_pk_id`, and the `ml_results` column, which uses `JSON` format:

  ```
  mysql> SELECT * FROM census_train_permutation LIMIT 3;
  +-------------------+-----+-----------+--------+-----------+---------------+--------------------+----------
  | _4aad19ca6e_pk_id | age | workclass | fnlwgt | education | education-num | marital-status     | occupatio
  +-------------------+-----+-----------+--------+-----------+---------------+--------------------+----------
  |                 1 |  37 | Private   |  99146 | Bachelors |            13 | Married-civ-spouse | Exec-mana
  |                 2 |  34 | Private   |  27409 | 9th       |             5 | Married-civ-spouse | Craft-rep
  |                 3 |  30 | Private   | 299507 | Assoc-acdm |           12 | Separated          | Other-ser
  +-------------------+-----+-----------+--------+-----------+---------------+--------------------+----------
  ```

See Also

- Generate Predictions Explanations for a Table

7.1.9 ML_SCORE

ML_SCORE scores a model by generating predictions using the feature columns in a labeled dataset as input and comparing the predictions to ground truth values in the target column of the labeled dataset. The dataset used with ML_SCORE should have the same feature columns as the dataset used to train the model but the data should be different. For example, you might reserve 20 to 30 percent of the labeled training data for scoring.

ML_SCORE returns a computed metric indicating the quality of the model.

ML_SCORE Syntax

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);

options: {
 JSON_OBJECT("key","value"[,"key","value"] ...)
      "key","value": {
      ['threshold', 'N']
      ['topk', 'N']
      ['remove_seen', {'true'|'false'}]
      }
}
```

Required ML_SCORE Parameters

Set the following required parameters.

- `table_name`: Specifies the fully qualified name of the table used to compute model quality (*database_name.table_name*). The table must contain the same columns as the training dataset.
- `target_column_name`: If scoring a supervised or semi-supervised model, specify the name of the target column containing ground truth values. If scoring an unsupervised model, set to `NULL`. See MySQL HeatWave AutoML Learning Types.
- `model_handle`: Specifies the model handle or a session variable containing the model handle. See Work with Model Handles.
- `metric`: Specifies the name of the metric. The metric selected must be compatible with the `task` type used for training the model. See Optimization and Scoring Metrics.
- `score`: Specifies the user-defined variable name for the computed score. The ML_SCORE routine populates the variable. User variables are written as *@var_name*. Any valid name for a user-defined variable is permitted.

The following options in `JSON` format are available for recommendation and anomaly detection models.

Options for Recommendation Models

Set the following options as needed for recommendation models.

- `threshold`: The optional threshold that defines positive feedback, and a relevant sample. Only use with ranking metrics. It can be used for either explicit or implicit feedback.
- `topk`: The optional top number of recommendations to provide. The default is `3`. Set a positive integer between 1 and the number of rows in the table.

A `recommendation` task and ranking metrics can use both `threshold` and `topk`.

- `remove_seen`: If the input table overlaps with the training table, and `remove_seen` is `true`, then the model will not repeat existing interactions. The default is `true`. Set `remove_seen` to `false` to repeat existing interactions from the training table.

Options for Anomaly Detection Models

Set the following options as needed for anomaly detection models.

- `threshold`: The threshold you set on anomaly detection models determines which rows in the output table are labeled as anomalies with an anomaly score of `1`, or normal with an anomaly score of `0`. The value for the threshold is the degree to which a row of data or log segment is considered for anomaly detection. Any sample with an anomaly score above the threshold is classified an anomaly. The default value is (1 - `contamination`)-th percentile of all the anomaly scores.
- `topk`: The optional top K rows to display with the highest anomaly scores. Set a positive integer between 1 and the number of rows in the table. If `topk` is not set, `ML_SCORE` uses `threshold`.

  Do not set both `threshold` and `topk`. Use `threshold` or `topk`, or set *`options`* to `NULL`.

Syntax Example

- The following example runs generates a score by using the `balanced_accuracy` metric. Query the score with the session variable for the `ML_SCORE` routine.

```
mysql> CALL sys.ML_SCORE('census_data.census_train', 'revenue', 'census_data.census_train_admin_174543994517
Query OK, 0 rows affected (3.0536 sec)
mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.8151071071624756 |
+--------------------+
1 row in set (0.0411 sec)
```

- The following example uses the `accuracy` metric with a `threshold` set to 90%.

```
mysql> CALL sys.ML_SCORE('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', 'target', @anomaly, 'accur
Query OK, 0 rows affected (1.86 sec)

mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.9791129231452942 |
+--------------------+
1 row in set (0.00 sec)
```

- The following example uses the `precision_at_k` metric with a `topk` value of 10.

```
mysql> CALL sys.ML_SCORE('mlcorpus_anomaly_detection.volcanoes-b3_anomaly_train', 'target', @anomaly, 'preci
Query OK, 0 rows affected (5.84 sec)

mysql> SELECT @score;
+---------------------+
| @score              |
+---------------------+
| 0                   |
+---------------------+
1 row in set (0.00 sec)
```

- The following example overrides the `ensemble_score` value from the `ML_TRAIN` routine to a new value of 0.5.

```
mysql> CALL sys.ML_SCORE('mlcorpus.anomaly_train_with_target', "target", @semsup_gknn, 'precision_at_k',
        CAST('{"topk": 10, "experimental": {"semisupervised": {"supervised_submodel_weight": 0.5}}}' as 
```

See also:

See Also

- Score a Model

7.1.10 ML_MODEL_LOAD

The `ML_MODEL_LOAD` routine loads a model from the model catalog. A model remains loaded until the model is unloaded using the `ML_MODEL_UNLOAD` routine

Use `ML_MODEL_ACTIVE` to check which models are active for which users. All active users and models share the amount of memory defined by the shape, and it might be necessary to schedule users. `ML_MODEL_LOAD` generates an error if there are memory limitations.

ML_MODEL_LOAD Syntax

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, user);
```

ML_MODEL_LOAD Parameters

Set the following parameters.

- `model_handle`: Specifies the model handle or a session variable containing the model handle. To look up a model handle, see Query the Model Handle.
- `user`: The MySQL user name of the model owner. You can set this to `NULL`. To learn how to share models with other users, see Grant Other Users Access to a Model.

Syntax Examples

- An example that specifies the model handle and sets the `user` parameter to `NULL`.

```
mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', NULL);
```

- An example that specifies a session variable containing the model handle.

```
mysql> CALL sys.ML_MODEL_LOAD(@iris_model, NULL);
```

- An example that specifies the model handle and the model owner.

```
mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', user1);
```

See Also

- Load a Model
- Work with Model Handles

7.1.11 ML_MODEL_UNLOAD

`ML_MODEL_UNLOAD` unloads a model from MySQL HeatWave AutoML.

Note

ML_MODEL_UNLOAD does not check whether the model specified is in the model catalog. If it is not, ML_MODEL_UNLOAD will succeed, but will not unload any model. Use ML_MODEL_ACTIVE to check which models are active and owned by the user.

ML_MODEL_UNLOAD Syntax

```
mysql> CALL sys.ML_MODEL_UNLOAD(model_handle);
```

To run ML_MODEL_UNLOAD, define the model_handle. To look up a model handle, see Query the Model Handle.

Syntax Examples

- An example that specifies the model handle.

  ```
  mysql> CALL sys.ML_MODEL_UNLOAD('ml_data.iris_train_user1_1636729526');
  ```

- An example that specifies a session variable containing the model handle.

  ```
  mysql> CALL sys.ML_MODEL_UNLOAD(@iris_model);
  ```

7.1.12 ML_MODEL_ACTIVE

Use the ML_MODEL_ACTIVE routine to check which models are loaded and active for which users. All active users and models share the amount of memory defined by the shape, and it might be necessary to schedule users.

ML_MODEL_ACTIVE Syntax

```
mysql> CALL sys.ML_MODEL_ACTIVE (user, model_info);
```

ML_MODEL_ACTIVE parameters:

- user: The user to provide information for. Set to current or all or NULL. NULL is equivalent to current.
- model_info: The name of the JSON array session variable that contains the active user and model information. There are two JSON object literals.

  If user is set to current or NULL, the following information displays.

  - A JSON object literal that displays:
    - Key: The total model size (bytes).
    - Value: The sum of model sizes for the current user.
  - A second JSON object literal that displays:
    - Key: The model handle for a loaded and active model owned by the current user.
    - Value: The model_metadata for the model.

  If user is set to all, the following information displays.

  - A JSON object literal that displays:

    - Key: The total model size (bytes).
    - Value: The sum of model sizes for all users.
- A second JSON object literal that displays:
    - Key: The name of a user who has loaded and active models.
    - Value: A list of JSON object literals of the model handle and brief model metadata for each loaded and active model.

Syntax Examples

- *user1* checks their own models:

```
mysql> CALL sys.ML_MODEL_ACTIVE('current', @model_info);
Query OK, 0 rows affected (0.10 sec)

mysql> SELECT JSON_PRETTY(@model_info);
+-----------------------------------------------------------+
| JSON_PRETTY(@model_info)                                  |
+-----------------------------------------------------------+
| [                                                         |
{                                                         |
"total model size(bytes)": 348954                       |
},                                                        |
{                                                         |
"iris_export_user1": {                                  |
  "task": "classification",                             |
  "notes": "",                                          |
  "chunks": 1,                                          |
  "format": "HWMLv2.0",                                 |
  "n_rows": 120,                                        |
  "status": "Ready",                                    |
  "options": {                                          |
    "model_explainer": "permutation_importance, shap",  |
    "prediction_explainer": "shap"                      |
  },                                                    |
  "n_columns": 4,                                       |
  "pos_class": null,                                    |
  "column_names": [                                     |
    "sepal length",                                     |
    "sepal width",                                      |
    "petal length",                                     |
    "petal width"                                       |
  ],                                                    |
  "contamination": null,                                |
  "model_quality": "high",                              |
  "training_time": 18.363686,                           |
  "algorithm_name": "ExtraTreesClassifier",             |
  "training_score": -0.10970368035588404,               |
  "build_timestamp": 1697524180,                        |
  "n_selected_rows": 96,                                |
  "training_params": {                                  |
    "sp_arr": null,                                     |
    "timezone": null,                                   |
    "recommend": "ratings",                             |
    "force_use_X": false,                               |
    "recommend_k": 3,                                   |
    "remove_seen": true,                                |
    "contamination": null,                              |
    "feedback_threshold": 1                             |
  },                                                    |
  "train_table_name": "mlcorpus.iris_train",            |
```

```
  "model_explanation": {                                        |
    "shap": {                                                   |
      "petal width": 0.3139,                                    |
      "sepal width": 0.0296,                                    |
      "petal length": 0.2787,                                   |
      "sepal length": 0.0462                                    |
    },                                                          |
    "permutation_importance": {                                 |
      "petal width": 0.2301,                                    |
      "sepal width": 0.0056,                                    |
      "petal length": 0.2192,                                   |
      "sepal length": 0.0056                                    |
    }                                                           |
  },                                                            |
  "model_object_size": 348954,                                  |
  "n_selected_columns": 4,                                      |
  "target_column_name": "class",                                |
  "optimization_metric": "neg_log_loss",                        |
  "selected_column_names": [                                    |
    "petal length",                                             |
    "petal width",                                              |
    "sepal length",                                             |
    "sepal width"                                               |
  ]                                                             |
}                                                               |
}                                                                 |
] |                                                                 |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

- *user1* checks their own models, and extracts specific information:

```
mysql> CALL sys.ML_MODEL_ACTIVE('current', @model_info);
Query OK, 0 rows affected (0.12 sec)

mysql> SELECT JSON_KEYS(JSON_EXTRACT(@model_info, '$[1]'));
+--------------------------------------------------------------+
| JSON_KEYS(JSON_EXTRACT(@model_info, '$[1]'))                 |
+--------------------------------------------------------------+
| ["iris_export", "mlcorpus.iris_train_user1_1697524152037"] |
+--------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_EXTRACT(@model_info, '$[0]');
+----------------------------------------+
| JSON_EXTRACT(@model_info, '$[0]')      |
+----------------------------------------+
| {"total model size(bytes)":  697908} |
+----------------------------------------+
1 row in set (0.01 sec)
```

- *user1* checks the models for all users:

```
mysql> CALL sys.ML_MODEL_ACTIVE('all', @model_info);
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT JSON_PRETTY(@model_info);
+-----------------------------------------------------+
| JSON_PRETTY(@model_info)                            |
+-----------------------------------------------------+
| [                                                   |
{                                                   |
"total model size(bytes)": 1046862                |
},                                                  |
{                                                   |
"user2": [                                        |
  {                                               |
```

```
    "iris_export_user2": {                          |
      "format": "HWMLv2.0",                         |
      "model_size(byte)": 348954                    |
    }                                               |
  }                                                 |
],                                                  |
"user1": [                                          |
  {                                                 |
    "mlcorpus.iris_train_user1_1697524152037": {    |
      "format": "HWMLv2.0",                         |
      "model_size(byte)": 348954                    |
    }                                               |
  },                                                |
  {                                                 |
    "iris_export": {                                |
      "format": "HWMLv2.0",                         |
      "model_size(byte)": 348954                    |
    }                                               |
  }                                                 |
]                                                   |
}                                                     |
] |                                                     |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

7.1.13 Model Types

MySQL HeatWave AutoML supports the following training models. When training a model, use the `ML_TRAIN` `model_list` and `exclude_model_list` options to specify the training models to consider or exclude. The Model Metadata includes the `algorithm_name` field, which defines the model type.

Classification Models

- LogisticRegression
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- XGBClassifier
- LGBMClassifier
- SVC
- LinearSVC
- ExtraTreesClassifier

Regression Models

- DecisionTreeRegressor
- RandomForestRegressor
- LinearRegression
- LGBMRegressor
- XGBRegressor

- SVR
- LinearSVR
- ExtraTreesRegressor

Forecasting Models

Univariate endogenous models:

- NaiveForecaster
- ThetaForecaster
- ExpSmoothForecaster
- ETSForecaster
- `STLwESForecaster`: STLForecast with ExponentialSmoothing substructure
- `STLwARIMAForecaster`: STLForecast with ARIMA substructure

Univariate endogenous with exogenous models:

- SARIMAXForecaster
- OrbitForecaster

Multivariate endogenous with exogenous models:

- VARMAXForecaster

Univariate or multivariate endogenous with exogenous models:

- DynFactorForecaster

Anomaly Detection Models

- GkNN: Generalized kth Nearest Neighbors
- PCA: Principal Component Analysis
- GLOF: Generalized Local Outlier Factor

Recommendation Models

Recommendation models that rate users or items to use with explicit feedback:

- Baseline
- CoClustering
- NormalPredictor
- SlopeOne
- Matrix factorization models:
  - SVD
  - SVDpp

- NMF

Recommendation models that rank users or items to use with implicit feedback:

- BPR: Bayesian Personalized Ranking from Implicit Feedback
- CTR: Collaborative Topic Regression

7.1.14 Optimization and Scoring Metrics

The `ML_TRAIN` routine includes the `optimization_metric` option, and the `ML_SCORE` routine includes the `metric` option. Both of these options define a metric that must be compatible with the `task` type and the target data. Model Metadata includes the `optimization_metric` field.

For more information about scoring metrics, see: scikit-learn.org. For more information about forecasting metrics, see: sktime.org and statsmodels.org.

Classification Metrics

Binary-only metrics:

- f1
- precision
- recall
- roc_auc

Binary and multi-class metrics:

- accuracy
- balanced_accuracy (`ML_SCORE` only)
- f1_macro
- f1_micro
- f1_samples (`ML_SCORE` only)
- f1_weighted
- neg_log_loss
- precision_macro
- precision_micro
- precision_samples (`ML_SCORE` only)
- precision_weighted
- recall_macro
- recall_micro
- recall_samples (`ML_SCORE` only)
- recall_weighted

Regression Metrics

- neg_mean_absolute_error
- neg_mean_squared_error
- neg_mean_squared_log_error
- neg_median_absolute_error
- r2

Forecasting Metrics

- neg_max_absolute_error
- neg_mean_absolute_error
- neg_mean_abs_scaled_error
- neg_mean_squared_error
- neg_root_mean_squared_error
- neg_root_mean_squared_percent_error
- neg_sym_mean_abs_percent_error

Anomaly Detection Metrics

Metrics for anomaly detection can only be used with the `ML_SCORE` routine. They cannot be used with the `ML_TRAIN` routine.

- roc_auc: You must not specify `threshold` or `topk` options.
- `precision_k`: An Oracle implementation of a common metric for fraud detection and lead scoring. You must use the `topk` option. You cannot use the `threshold` option.

The following metrics can use the `threshold` option, but cannot use the `topk` option:

- accuracy
- balanced_accuracy
- f1
- neg_log_loss
- precision
- recall

Recommendation Model Metrics

The following rating metrics can be used for explicit feedback:

- neg_mean_absolute_error
- neg_mean_squared_error
- neg_root_mean_squared_error

- r2

For recommendation models that use implicit feedback:

- If a user and item combination in the input table is not unique, the input table is grouped by user and item columns, and the result is the average of the rankings.
- If the input table overlaps with the training table, and `remove_seen` is `true`, which is the default setting, then the model will not repeat a recommendation and it ignores the overlap items.

The following ranking metrics can be used for implicit and explicit feedback:

- `precision_at_k` is the number of relevant `topk` recommended items divided by the total `topk` recommended items for a particular user:

  `precision_at_k` = (relevant `topk` recommended items) / (total `topk` recommended items)

  For example, if 7 out of 10 items are relevant for a user, and `topk` is 10, then `precision_at_k` is 70%.

  The `precision_at_k` value for the input table is the average for all users. If `remove_seen` is `true`, the default setting, then the average only includes users for whom the model can make a recommendation. If a user has implicitly ranked every item in the training table, the model cannot recommend any more items for that user, and they are ignored from the average calculation if `remove_seen` is `true`.

- `recall_at_k` is the number of relevant `topk` recommended items divided by the total relevant items for a particular user:

  `recall_at_k` = (relevant `topk` recommended items) / (total relevant items)

  For example, there is a total of 20 relevant items for a user. If `topk` is 10, and 7 of those items are relevant, then `recall_at_k` is 7 / 20 = 35%.

  The `recall_at_k` value for the input table is the average for all users.

- `hit_ratio_at_k` is the number of relevant `topk` recommended items divided by the total relevant items for all users:

  `hit_ratio_at_k` = (relevant `topk` recommended items, all users) / (total relevant items, all users)

  The average of `hit_ratio_at_k` for the input table is `recall_at_k`. If there is only one user, `hit_ratio_at_k` is the same as `recall_at_k`.

- `ndcg_at_k` is normalized discounted cumulative gain, which is the discounted cumulative gain of the relevant `topk` recommended items divided by the discounted cumulative gain of the relevant `topk` items for a particular user.

  The discounted gain of an item is the true rating divided by $\log_2(r+1)$ where `r` is the ranking of this item in the relevant `topk` items. If a user prefers a particular item, the rating is higher, and the ranking is lower.

  The `ndcg_at_k` value for the input table is the average for all users.

7.2 GenAI Routines

MySQL HeatWave GenAI routines reside in the MySQL `sys` schema.

MySQL JavaScript Stored Programs include a GenAI API that you can use to call different MySQL HeatWave GenAI routines using JavaScript functions. For more information, see JavaScript GenAI API.

7.2.1 ML_GENERATE

The `ML_GENERATE` routine uses the specified large language model (LLM) to generate text-based content as a response for the given natural-language query.

This topic contains the following sections:

- ML_GENERATE Syntax
- Syntax Examples
- See Also

ML_GENERATE Syntax

```
mysql> SELECT sys.ML_GENERATE('QueryInNaturalLanguage'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'task', {'generation'|'summarization'}
  |'model_id', 'LargeLanguageModelID'
  |'context', 'Context'
  |'language', 'Language'
  |'temperature', Temperature
  |'max_tokens', MaxTokens
  |'top_k', K
  |'top_p', P
  |'repeat_penalty', RepeatPenalty
  |'frequency_penalty', FrequencyPenalty
  |'stop_sequences', JSON_ARRAY('StopSequence'[, 'StopSequence'] ...)
  |'speculative_decoding', {true|false}
}
```

Following are `ML_GENERATE` parameters:

- *`QueryInNaturalLanguage`*: specifies the natural-language query that is passed to the large language model (LLM) handle.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `task`: specifies the task expected from the LLM. Default value is `generation`. Possible values are:
    - `generation`: generates text-based content.
    - `summarization`: generates a summary for existing text-based content.
  - `model_id`: specifies the LLM to use for the task. Default and possible value is `llama3.2-3b-instruct-v1`.

    To view the lists of available LLMs, see In-Database LLM.
  - `context`: specifies the context to be used for augmenting the query and guide the text generation of the LLM. Default value is `NULL`.
  - `language`: specifies the language to be used for writing queries, ingesting documents, and generating the output. To set the value of the `language` parameter, use the two-letter `ISO 639-1` code for the language.

    Default value is `en`.

For possible values, to view the list of supported languages, see Section 5.4, “Supported LLM, Embedding Model, and Languages”.

- `temperature`: specifies a non-negative float that tunes the degree of randomness in generation. Lower temperatures mean less random generations.

  Default value is `0` for all LLMs.

  Possible values are float values between `0` and `5` for the In-Database LLM.

  It is suggested that:

  - To generate the same output for a particular prompt every time you run it, set the temperature to `0`.
  - To generate a random new statement for a particular prompt every time you run it, increase the temperature.

- `max_tokens`: specifies the maximum number of tokens to predict per generation using an estimate of three tokens per word. Default value is `256`. Possible values are integer values between`1` and `4096`.

- `top_k`: specifies the number of top most likely tokens to consider for text generation at each step. Default value is `40`, which means that top 40 most likely tokens are considered for text generation at each step. Possible values are integer values between `0` and `32000`.

- `top_p`: specifies a number, `p`, and ensures that only the most likely tokens with the sum of probabilities `p` are considered for generation at each step. A higher value of `p` introduces more randomness into the output. Default value is `0.95`. Possible values are float values between `0` and `1`.

  - To disable this method, set to `1.0` or `0`.
  - To eliminate tokens with low likelihood, assign `p` a lower value. For example, if set to `0.1`, tokens within top 10% probability are included.
  - To include tokens with low likelihood, assign `p` a higher value. For example, if set to `0.9`, tokens within top 90% probability are included.

  If you are also specifying the `top_k` parameter, the LLM considers only the top tokens whose probabilities add up to `p` percent. It ignores the rest of the `k` tokens.

- `repeat_penalty`: assigns a penalty when a token appears repeatedly. High penalties encourage less repeated tokens and produce more random outputs. Default value is `1.1`. Possible values are float values between `0` and `2`.

- `frequency_penalty`: assigns a penalty when a token appears frequently. High penalties encourage less repeated tokens and produce more random outputs. Default value is `0`. Possible values are float values between `0` and `1`.

- `stop_sequences`: specifies a list of characters such as a word, a phrase, a newline, or a period that tells the LLM when to end the generated output. If you have more than one stop sequence, then the LLM stops when it reaches any of those sequences. Default value is `NULL`.

Syntax Examples

- Generating text-based content in English using the `llama3.2-3b-instruct-v1` model:

```
mysql> SELECT sys.ML_GENERATE("What is AI?", JSON_OBJECT("task", "generation", "model_id", "llama3.2-3b-
```

- Summarizing English text using the `llama3.2-3b-instruct-v1` model:

```
mysql> SELECT sys.ML_GENERATE(@text, JSON_OBJECT("task", "summarization", "model_id", "llama3.2-3b-instruct-
```

  Where, `@text` is set as shown below:

```
SET @text="Artificial Intelligence (AI) is a rapidly growing field that has the potential to
revolutionize how we live and work. AI refers to the development of computer systems that can
perform tasks that typically require human intelligence, such as visual perception, speech
recognition, decision-making, and language translation.\n\nOne of the most significant developments in
AI in recent years has been the rise of machine learning, a subset of AI that allows computers to learn
from data without being explicitly programmed. Machine learning algorithms can analyze vast amounts
of data and identify patterns, making them increasingly accurate at predicting outcomes and making
decisions.\n\nAI is already being used in a variety of industries, including healthcare, finance, and
transportation. In healthcare, AI is being used to develop personalized treatment plans for patients
based on their medical history and genetic makeup. In finance, AI is being used to detect fraud and make
investment recommendations. In transportation, AI is being used to develop self-driving cars and improve
traffic flow.\n\nDespite the many benefits of AI, there are also concerns about its potential impact on
society. Some worry that AI could lead to job displacement, as machines become more capable of performing
tasks traditionally done by humans. Others worry that AI could be used for malicious ";
```

See Also

- Section 5.5, "Generating Text-Based Content"
- Section 7.2.2, "ML_GENERATE_TABLE"

7.2.2 ML_GENERATE_TABLE

The `ML_GENERATE_TABLE` routine runs multiple text generation or summarization queries in a batch, in parallel. The output generated for every input query is the same as the output generated by the `ML_GENERATE` routine.

This topic contains the following sections:

- ML_GENERATE_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 5.3, "Required Privileges for using GenAI".

ML_GENERATE_TABLE Syntax

```
mysql> CALL sys.ML_GENERATE_TABLE('InputTableColumn', 'OutputTableColumn'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'task', {'generation'|'summarization'}
  |'model_id', 'LargeLanguageModelID'
  |'context_column', 'ContextColumn'
  |'language', 'Language'
  |'temperature', Temperature
  |'max_tokens', MaxTokens
  |'top_k', K
  |'top_p', P
  |'repeat_penalty', RepeatPenalty
  |'frequency_penalty', FrequencyPenalty
  |'stop_sequences', JSON_ARRAY('StopSequence'[, 'StopSequence'] ...)
  |'batch_size', BatchSize
```

```
  |'speculative_decoding', {true|false}
}
```

Following are `ML_GENERATE_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the natural-language queries. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.
  - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated text-based response is stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified output table must be an internal table.
  - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new JSON column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.
  - If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a JSON column that stores the generated text-based responses.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `task`: specifies the task expected from the large language model (LLM). Default value is `generation`. Possible values are:
    - `generation`: generates text-based content.
    - `summarization`: generates a summary for existing text-based content.
  - `model_id`: specifies the LLM to use for the task. Default and possible value is `llama3.2-3b-instruct-v1`.

    To view the lists of available LLMs, see In-Database LLM.
  - `context_column`: specifies the table column that contains the context to be used for augmenting the queries and guiding the text generation of the LLM. The specified column must be an existing column in the input table. Default value is `NULL`.

- `language`: specifies the language to be used for writing queries, ingesting documents, and generating the output. To set the value of the `language` parameter, use the two-letter `ISO 639-1` code for the language.

  Default value is `en`.

  For possible values, to view the list of supported languages, see Section 5.4, "Supported LLM, Embedding Model, and Languages".

- `temperature`: specifies a non-negative float that tunes the degree of randomness in generation. Lower temperatures mean less random generations.

  Default value is `0` for all LLMs.

  Possible values are float values between `0` and `5` For the In-Database LLM.

  It is suggested that:

  - To generate the same output for a particular prompt every time you run it, set the temperature to `0`.

  - To generate a random new statement for a particular prompt every time you run it, increase the temperature.

- `max_tokens`: specifies the maximum number of tokens to predict per generation using an estimate of three tokens per word. Default value is `256`. Possible values are integer values between`1` and `4096`.

- `top_k`: specifies the number of top most likely tokens to consider for text generation at each step. Default value is `40`, which means that top 40 most likely tokens are considered for text generation at each step. Possible values are integer values between `0` and `32000`.

- `top_p`: specifies a number, `p`, and ensures that only the most likely tokens with the sum of probabilities `p` are considered for generation at each step. A higher value of `p` introduces more randomness into the output. Default value is `0.95`. Possible values are float values between `0` and `1`.

  - To disable this method, set to `1.0` or `0`.

  - To eliminate tokens with low likelihood, assign `p` a lower value. For example, if set to `0.1`, tokens within top 10% probability are included.

  - To include tokens with low likelihood, assign `p` a higher value. For example, if set to `0.9`, tokens within top 90% probability are included.

  If you are also specifying the `top_k` parameter, the LLM considers only the top tokens whose probabilities add up to `p` percent. It ignores the rest of the `k` tokens.

- `repeat_penalty`: assigns a penalty when a token appears repeatedly. High penalties encourage less repeated tokens and produce more random outputs. Default value is `1.1`. Possible values are float values between `0` and `2`.

- `frequency_penalty`: assigns a penalty when a token appears frequently. High penalties encourage less repeated tokens and produce more random outputs. Default value is `0`. Possible values are float values between `0` and `1`.

- `stop_sequences`: specifies a list of characters such as a word, a phrase, a newline, or a period that tells the LLM when to end the generated output. If you have more than one stop sequence, then the LLM stops when it reaches any of those sequences. Default value is `NULL`.

- batch_size: specifies the batch size for the routine. This option is supported for internal tables only. Default value is 1000. Possible values are integer values between 1 and 1000.

Syntax Examples

Generate English text-based content in a batch using the llama3.2-3b-instruct-v1 model for queries stored in demo_db.input_table:

```
mysql> CALL sys.ML_GENERATE_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OBJECT('
```

See Also

- Generate New Text - Run Batch Queries
- Summarize Content - Run Batch Queries
- Section 7.2.1, “ML_GENERATE”

7.2.3 VECTOR_STORE_LOAD

The VECTOR_STORE_LOAD routine generates vector embedding for the specified files or folders that are , and loads the embeddings into a new vector store table.

This topic contains the following sections:

- VECTOR_STORE_LOAD Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 5.3, “Required Privileges for using GenAI”.

VECTOR_STORE_LOAD Syntax

```
mysql> CALL sys.VECTOR_STORE_LOAD('URI'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'format', 'Format'
  |'schema_name', 'SchemaName'
  |'table_name', 'TableName'
  |'language', 'Language'
  |'embed_model_id', 'ModelID'
  |'description', 'Description'
  |'ocr', {true|false}
}
```

Following are VECTOR_STORE_LOAD parameters:

- *URI*: specifies the unique reference index (URI) of the files or folders to be ingested into the vector store.

  A URI is considered to be one of the following:

  - A glob pattern, if it contains at least one unescaped ? or * character.
  - A prefix, if it is not a pattern and ends with a / character like a folder path.
  - A file path, if it is neither a glob pattern nor a prefix.

- options: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - format: specifies the format of files to be loaded. Default value is auto_unstructured, which means all supported types of files are loaded. Possible values are pdf, pptx, ppt, txt, html, docx, doc, and auto_unstructured.
  - schema_name: specifies the name of the schema where the vector embeddings are to be loaded. By default, this procedure uses the current schema from the session.
  - table_name: specifies the name of the vector store table to create. By default, the routine generates a unique table name with format vector_store_data_x, where x is a counter.
  - language: specifies the text content language used in the files to be ingested into the vector store. To set the value of the language parameter, use the two-letter ISO 639-1 code for the language.

    Default value is en.

    For possible values, to view the list of supported languages, see Section 5.4, "Supported LLM, Embedding Model, and Languages".
  - embed_model_id: specifies the embedding model to use for encoding the text. Default value is multilingual-e5-small.

    For possible values, to view the list of available embedding models, see In-Database Embedding Model.
  - description: specifies a description of document collection being loaded. Default value is NULL.
  - ocr: specifies whether to enable or disable Optical Character Recognition (OCR). If set to false, disables OCR. Default value is true, which means OCR is enabled by default. Default value is true.

Syntax Examples

- Specifying the file to ingest, using the current database, auto-generated name for the vector store table, and default values for all options:

```
mysql> CALL sys.VECTOR_STORE_LOAD('file:///var/lib/mysql-files/demo-directory/heatwave-en.pdf', NULL);
```

- Specifying the file to ingest, using the current database, and specifying the name of the vector store table to be created:

```
mysql> CALL sys.VECTOR_STORE_LOAD('file:///var/lib/mysql-files/demo-directory/heatwave-en.pdf', '{"table_nam
```

- Specifying additional options such the schema name, table name, language, format, and table description in VECTOR_STORE_LOAD:

```
mysql> CALL sys.VECTOR_STORE_LOAD('file:///var/lib/mysql-files/german_files/de*', '{"schema_name": "demo_db"
```

See Also

Ingesting Files into a Vector Store

7.2.4 ML_RAG

The ML_RAG routine performs retrieval-augmented generation (RAG) by:

1. Taking a natural-language query.

2. Retrieving context from relevant documents using semantic search.

3. Generating a response that integrates information from the retrieved documents.

This routine aims to provide detailed, accurate, and contextually relevant answers by augmenting a generative model with information retrieved from a comprehensive knowledge base.

This topic contains the following sections:

- ML_RAG Syntax
- Syntax Examples
- See Also

ML_RAG Syntax

```
mysql> CALL sys.ML_RAG('QueryInNaturalLanguage', 'Output'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'vector_store', JSON_ARRAY('VectorStoreTableName'[, 'VectorStoreTableName']...)
  |'schema', JSON_ARRAY('SchemaName'[, 'SchemaName']...)
  |'n_citations', NumberOfCitations
  |'distance_metric', {'COSINE'|'DOT'|'EUCLIDEAN'}
  |'document_name', JSON_ARRAY('DocumentName'[, 'DocumentName']...)
  |'skip_generate', {true|false}
  |'model_options', modeloptions
  |'exclude_vector_store', JSON_ARRAY('ExcludeVectorStoreTableName'[, 'ExcludeVectorStoreTableName']...)
  |'exclude_document_name', JSON_ARRAY('ExcludeDocumentName'[, 'ExcludeDocumentName']...)
  |'retrieval_options', retrievaloptions
  |'vector_store_columns', vscoptions
  |'embed_model_id', 'EmbeddingModelID'
  |'query_embedding', 'QueryEmbedding'
}
```

Following are `ML_RAG` parameters:

- *`QueryInNaturalLangugae`*: specifies the natural-language query.
- *`Output`*: stores the generated output. The output contains the following segments:
  - `text`: the generated text-based response.
  - `citations`: contains the following details:
    - `segment`: the textual content that is retrieved from the vector store through semantic search, and used as context generating the response.
    - `distance`: the distance between the query embedding the segment embedding.
    - `document_name`: the name of the document from which the segment is retrieved.
  - `vector_store`: the list of vector store tables used for context retrieval.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `vector_store`: specifies a list of loaded vector store tables to use for context retrieval. The routine ignores invalid table names. By default, the routine performs a global search across all the available vector store tables in the DB system.

- `schema`: specifies a list of schemas to check for loaded vector store tables. By default, the routine performs a global search across all the available vector store tables in all the schemas that are available in the DB system.
- `n_citations`: specifies the number of segments to consider for context retrieval. Default value is `3`. Possible values are integer values between `0` and `100`.
- `distance_metric`: specifies the distance metrics to use for context retrieval. Default value is `COSINE`. Possible values are `COSINE`, `DOT`, and `EUCLIDEAN`.
- `document_name`: limits the documents to use for context retrieval. Only the specified documents are used. By default, the routine performs a global search across all the available documents stored in all the available vector stores in the DB system.
- `skip_generate`: specifies whether to skip generation of the text-based response, and only perform context retrieval from the available or specified vector stores, schemas, or documents. Default value is `false`.
- `model_options`: additional options that you can set for generating the text-based response. These are the same options that are available in the `ML_GENERATE` routine, which alter the text-based response per the specified settings. Default value is `'{"model_id": "llama3.2-3b-instruct-v1"}'`. To view the list of supported models, see Section 5.4, "Supported LLM, Embedding Model, and Languages".

  However, the `context` model option is not supported as an `ML_RAG` model option.
- `exclude_vector_store`: specifies a list of loaded vector store tables to exclude from context retrieval. The routine ignores invalid table names. Default value is `NULL`.
- `exclude_document_name`: specifies a list of documents to exclude from context retrieval. Default value is `NULL`.
- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  It can include the following parameters:

```
retrievaloptions: JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine fails to run.

> **Note**
>
> If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

Default value is `0.6` for all distance metrics.

Possible values are decimal values between `0` and `999999.9999`.

- `percentage_distance`: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  `MaximumDistanceThreshold` = *`DistanceOfInputQueryToNearestSegment`* + [(`percentage_distance` / `100`) * *`DistanceOfInputQueryToNearestSegment`*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  > **Note**
  >
  > If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

  Default value is `20` for all distance metrics.

  Possible values are decimal values between `0` and `999999.9999`.

  > **Note**
  >
  > If both `max_distance` and `percentage_distance` are set, the smaller threshold value is considered for filtering out the segments.

- `segment_overlap`: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is `1`. Possible values are integer values between `0` and `5`.

- `vector_store_columns`: specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

  It can include the following parameters:

```
vscoptions: JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsckeyv
vsckeyvalue:
{
  'document_name', 'DocumentName'
  |'document_id', DocumentID
  |'metadata', 'Metadata'
  |'segment_number', SegmentNumber
```

```
}
```

  - `segment`: specifies the name of the mandatory string column that contains the text segments. Default value is `segment`.
  - `segment_embedding`: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is `segment_embedding`.
  - `document_name`: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL. Default value is `document_name`.
  - `document_id`: specifies the name of the optional integer column that contains the document IDs. Default value is `document_id`.
  - `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
  - `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

  Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

- `embed_model_id`: specifies the embedding model to use for embedding the input query. If you are providing the query embedding, then set this parameter to specify the embedding model to use to embed the query. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval. Default value is `multilingual-e5-small`.

  To view the list of available embedding models, see In-Database Embedding Model.

- `query_embedding`: specifies the vector embedding of the input query. If this parameter is set, then the routine skips generating the vector embeddings of the input query. Instead, it uses this embedding for context retrieval from valid vector store and embedding tables that contain vector embeddings created using the same embedding model.

Syntax Examples

Retrieving context and generating output:

```
mysql> CALL sys.ML_RAG("What is AutoML",@output,@options);
```

Where, `@options` is set to specify the vector store table to use using `vector_store` key, as shown below:

```
mysql> SET @options = JSON_OBJECT("vector_store", JSON_ARRAY("demo_db.demo_embeddings"));
```

See Also

- Section 5.8, “Performing Vector Search with Retrieval-Augmented Generation”
- Section 7.2.5, “ML_RAG_TABLE”

7.2.5 ML_RAG_TABLE

The `ML_RAG_TABLE` routine runs multiple retrieval-augmented generation (RAG) queries in a batch, in parallel. The output generated for every input query is the same as the output generated by the `ML_RAG` routine.

This topic contains the following sections:

- ML_RAG_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 5.3, "Required Privileges for using GenAI".

ML_RAG_TABLE Syntax

```
mysql> CALL sys.ML_RAG_TABLE('InputTableColumn', 'OutputTableColumn'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue]...)
keyvalue:
{
  'vector_store', JSON_ARRAY('VectorStoreTableName'[, 'VectorStoreTableName']...)
  |'schema', JSON_ARRAY('SchemaName'[, 'SchemaName']...)
  |'n_citations', NumberOfCitations
  |'distance_metric', {'COSINE'|'DOT'|'EUCLIDEAN'}
  |'document_name', JSON_ARRAY('DocumentName'[, 'DocumentName']...)
  |'skip_generate', {true|false}
  |'model_options', modeloptions
  |'exclude_vector_store', JSON_ARRAY('ExcludeVectorStoreTableName'[, 'ExcludeVectorStoreTableName']...)
  |'exclude_document_name', JSON_ARRAY('ExcludeDocumentName'[, 'ExcludeDocumentName']...)
  |'batch_size', BatchSize
  |'retrieval_options', retrievaloptions
  |'vector_store_columns', vscoptions
  |'embed_model_id', 'EmbeddingModelID'
  |'embed_column', 'EmbeddedQueriesColumnName'
  |'fail_on_embedding_error', {true|false}
}
```

Following are `ML_RAG_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the natural-language queries. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.
  - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.
- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated text-based response is stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.

  - The specified output table must be an internal table.
  - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new JSON column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.
  - If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a JSON column that stores the generated text-based responses.
  - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.

- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `vector_store`: specifies a list of loaded vector store tables to use for context retrieval. The routine ignores invalid table names. By default, the routine performs a global search across all the available vector store tables in the DB system.
  - `schema`: specifies a list of schemas to check for loaded vector store tables. By default, the routine performs a global search across all the available vector store tables in all the schemas that are available in the DB system.
  - `n_citations`: specifies the number of segments to consider for context retrieval. Default value is `3`. Possible values are integer values between `0` and `100`.
  - `distance_metric`: specifies the distance metrics to use for context retrieval. Default value is `COSINE`. Possible values are `COSINE`, `DOT`, and `EUCLIDEAN`.
  - `document_name`: limits the documents to use for context retrieval. Only the specified documents are used. By default, the routine performs a global search across all the available documents stored in all the available vector stores in the DB system.
  - `skip_generate`: specifies whether to skip generation of the text-based response, and only perform context retrieval from the available or specified vector stores, schemas, or documents. Default value is `false`.
  - `model_options`: additional options that you can set for generating the text-based response. These are the same options that are available in the `ML_GENERATE` routine, which alter the text-based response per the specified settings. However, the `context` option is not supported as an

ML_RAG_TABLE model option. Default value is `'{"model_id": "llama3.2-3b-instruct-v1"}'`.

- `exclude_vector_store`: specifies a list of loaded vector store tables to exclude from context retrieval. The routine ignores invalid table names. Default value is `NULL`.
- `exclude_document_name`: specifies a list of documents to exclude from context retrieval. Default value is `NULL`.
- `batch_size`: specifies the batch size for the routine. This option is supported for internal tables only. Default value is `1000`. Possible values are integer values between `1` and `1000`.
- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  It can include the following parameters:

```
retrievaloptions: JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine generates an output without using any context.

    Note

    If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

    Default value is `0.6` for all distance metrics.

    Possible values are decimal values between `0` and `999999.9999`.

- percentage_distance: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  MaximumDistanceThreshold = *DistanceOfInputQueryToNearestSegment* + [(percentage_distance / 100) * *DistanceOfInputQueryToNearestSegment*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  Note

  If this parameter is set, the default value of the n_citations parameter is automatically updated to 10.

  Default value is 20 for all distance metrics.

  Possible values are decimal values between 0 and 999999.9999.

  Note

  If both max_distance and percentage_distance are set, the smaller threshold value is considered for filtering out the segments.

- segment_overlap: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is 1. Possible values are integer values between 0 and 5.

* vector_store_columns: specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

  It can include the following parameters:

```
vscoptions: JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsckeyvalue
vsckeyvalue:
{
  'document_name', 'DocumentName'
  |'document_id', DocumentID
  |'metadata', 'Metadata'
  |'segment_number', SegmentNumber
}
```

  - segment: specifies the name of the mandatory string column that contains the text segments. Default value is segment.

  - segment_embedding: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is segment_embedding.

  - document_name: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL. Default value is document_name.

  - document_id: specifies the name of the optional integer column that contains the document IDs. Default value is document_id.

  - `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
  - `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

  Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

- `embed_model_id`: specifies the embedding model to use for embedding the input queries. If you are providing the query embeddings, then set this option to specify the embedding model to use to embed the queries. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval. Default value is `multilingual-e5-small`.

  To view the list of available embedding models, see In-Database Embedding Model.

- `embed_column`: specifies the name of the input table column which contains vector embeddings of the input queries. If this option is set, then the routine skips generating the vector embeddings of the input queries. Instead, it uses the embeddings stored in this column for context retrieval from valid vector store and embedding tables that contain vector embeddings created using the same embedding model.

- `fail_on_embedding_error`: if set to `true`, stops the batch processing of input queries and throws an error in case an error is encountered for an input row. If set to `false`, allows the batch processing to partially fail for rows where errors are encountered, and lets the routine continue with processing the other rows. Default value is `true`.

Syntax Examples

Running retrieval-augmented generation in a batch of 10:

```
mysql> CALL sys.ML_RAG_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OBJECT("vecto
```

In this example, the routine performs RAG for 10 input queries stored in the `demo_db.input_table.Input` column, and creates a column of 10 rows `demo_db.output_table.Output` where it stores the generated outputs.

See Also

- Run Retrieval-Augmented Generation - Run Batch Queries
- Use Your Own Embeddings with Retrieval-Augmented Generation - Run Batch Queries
- Section 7.2.4, “ML_RAG”

7.2.6 HEATWAVE_CHAT

The `HEATWAVE_CHAT` routine automatically calls the `ML_RAG` routine which loads an LLM and runs a semantic search on the available vector stores by default. If the routine cannot find a vector store, then it calls the `ML_GENERATE` routine and uses information available in LLM training data, which is primarily information that is available in public data sources, to generate a response for the entered query.

This topic contains the following sections:

- HEATWAVE_CHAT Syntax

- @chat_options Parameters
- Syntax Examples
- See Also

HEATWAVE_CHAT Syntax

```
mysql> CALL sys.HEATWAVE_CHAT('QueryInNaturalLanguage');
```

The `HEATWAVE_CHAT` routine accepts one input parameter:

- *`QueryInNaturalLanguage`*: specifies the query in natural language.

For specifying additional chat parameter settings, the `HEATWAVE_CHAT` routine reserves a variable, `@chat_options`. When you run the routine, it also updates the `@chat_options` variable with any additional information that is used or collected by the routine to generate the response.

@chat_options Parameters

Following is a list of all the parameters that you can set in the `@chat_options` variable:

- **Input only**: you can set these parameters to control the chat behavior. The routine cannot change the values of these parameters.
  - `schema_name`: specifies the name of a schema. If set, the routine searches for vector store tables in this schema. This parameter cannot be used in combination with the `tables` parameter. Default value is `NULL`
  - `report_progress`: specifies whether information such as routine progress detail is to be reported. Default value is `false`.
  - `skip_generate`: specifies whether response generation is skipped. If set to `true`, the routine does not generate a response. Default value is `false`.
  - `return_prompt`: specifies whether to return the prompt that was passed to the `ML_RAG` or `ML_GENERATE` routines. Default value is `false`.
  - `re_run`: if set to `true`, it indicates that the request is a re-run of the previous request. For example, a re-run of a query with some different parameters. The new query and response replaces the last entry stored in the `chat_history` parameter. Default value is `false`.
  - `include_document_uris`: limits the documents used for context retrieval by including only the specified document URIs. Default value is `NULL`.
  - `retrieve_top_k`: specifies the context size. The default value is the value of the `n_citations` parameter of the `ML_RAG` routine. Possible values are integer values between `0` and `100`.
  - `chat_query_id`: specifies the chat query ID to be printed with the `chat_history` in the GUI. This parameter is reserved for GUI use. By default, the routine generates random IDs.
  - `history_length`: specifies the maximum history length, which is the number of question and answers, to include in the chat history. The specified value must be greater than or equal to `0`. Default value is `3`.
  - `vector_store_columns`: optional parameter which specifies column names for finding relevant vector and embedding tables for context retrieval as key-value pairs in JSON format. If multiple tables contain columns with the same name and data type, then all such tables are used for context retrieval.

It can include the following parameters:

```
JSON_OBJECT('segment', 'SegmentColName', 'segment_embedding', 'EmbeddingColName'[, vsckeyvalue]...)
vsckeyvalue:
{
  'document_name', 'DocumentName'
  |'document_id', DocumentID
  |'metadata', 'Metadata'
  |'segment_number', SegmentNumber
}
```

- `segment`: specifies the name of the mandatory string column that contains the text segments. Default value is `segment`.
- `segment_embedding`: specifies the name of the mandatory vector column that contains vector embeddings of the text segments. Default value is `segment_embedding`.
- `document_name`: specifies the name of the optional column that contains the document names. This column can be of any data type supported by MySQL. Default value is `document_name`.
- `document_id`: specifies the name of the optional integer column that contains the document IDs. Default value is `document_id`.
- `metadata`: specifies the name of the optional JSON column that contains additional table metadata. Default value is `metadata`.
- `segment_number`: specifies the name of the optional integer column that contains the segment numbers. Default value is `segment_number`.

Default value is `{"segment": "segment", "segment_embedding": "segment_embedding", "document_id: "document_id", "segment_number": "segment_number", "metadata": "metadata"}`, which means that by default, the routine uses the default values of all column names to find relevant tables for context retrieval.

- `embed_model_id`: specifies the embedding model to use for embedding the input query. The routine uses vector store tables and embedding tables created using the same embedding model for context retrieval. Default value is `multilingual-e5-small`.

  To view the list of available embedding models, see In-Database Embedding Model.

- `retrieval_options`: specifies optional context retrieval parameters as key-value pairs in JSON format. If a parameter value in `retrieval_options` is set to `auto`, the default value for that parameter is used.

  It can include the following parameters:

```
JSON_OBJECT(retrievaloptkeyvalue[, retrievaloptkeyvalue]...)
retrievaloptkeyvalue:
{
  'max_distance', MaxDistance
  |'percentage_distance', PercentageDistance
  |'segment_overlap', SegmentOverlap
}
```

  - `max_distance`: specifies a maximum distance threshold for filtering out segments from context retrieval. Segments for which the distance from the input query exceeds the specified maximum distance threshold are excluded from content retrieval. This ensures that only the segments that are closer to the input query are included during context retrieval. However, if no segments are found within the specified distance, the routine fails to run.

> **Note**
>
> If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

Default value is `0.6` for all distance metrics.

Possible values are decimal values between `0` and `999999.9999`.

- `percentage_distance`: specifies what percentage of distance to the nearest segment is to be used to determine the maximum distance threshold for filtering out segments from context retrieval.

  Following is the formula used for calculating the maximum distance threshold:

  `MaximumDistanceThreshold` = *`DistanceOfInputQueryToNearestSegment`* + [(`percentage_distance` / `100`) * *`DistanceOfInputQueryToNearestSegment`*]

  Which means that the segments for which the distance to the input query exceeds the distance of the input query to the nearest segment by the specified percentage are filtered out from context retrieval.

  > **Note**
  >
  > If this parameter is set, the default value of the `n_citations` parameter is automatically updated to `10`.

  Default value is `20` for all distance metrics.

  Possible values are decimal values between `0` and `999999.9999`.

  > **Note**
  >
  > If both `max_distance` and `percentage_distance` are set, the smaller threshold value is considered for filtering out the segments.

- `segment_overlap`: specifies the number of additional segments adjacent to the nearest segments to the input query to be included in context retrieval. These additional segments provide more continuous context for the input query. Default value is `1`. Possible values are integer values between `0` and `5`.

- **Input-output**: both you and the routine can change the values of these parameters.

  - `chat_history`: JSON array that represents the current chat history. Default value is `NULL`.

    Syntax for each object in the `chat_history` array is as follows:

```
JSON_OBJECT('key','value'[,'key','value'] ...)
  'key','value': {
  ['user_message','Message']
  ['chat_bot_message','Message']
  ['chat_query_id','ID']
```

```
}
```

Each parameter value in the array holds the following keys and their values:

- user_message: message entered by the user.
- chat_bot_message: message generated by the chat bot.
- chat_query_id: a query ID.

- tables: JSON array that represents the following:

  - For providing input, represents the list of vector store schema or table names to consider for context retrieval.
  - As routine output, represents the list of discovered vector store tables, if any. Otherwise, it holds the same values as input.

  Default value is NULL.

  Syntax for each object in the tables array is as follows:

```
JSON_OBJECT('key','value'[,'key','value'] ...)
  'key','value': {
  ['schema_name','SchemaName']
  ['table_name','TableName']
}
```

  Each parameter values in the array holds the following keys and their values:

  - schema_name: name of the schema.
  - table_name: name of the vector store table.

- task: specifies the task performed by the LLM. Default value is generation. Possible value is generation.

- model_options: optional model parameters specified as key-value pairs in JSON format. These are the same options that are available in the ML_GENERATE routine, which alter the text-based response per the specified settings. Default value is '{"model_id": "llama3.2-3b-instruct-v1"}'.

- **Output only**: only the routine can set or change values of these parameters.
  - `info`: contains information messages such as routine progress information. Default value is `NULL`. This parameter is populated only if `report_progress` is set to `true`.
  - `error`: contains the error message if an error occurred. Default value is `NULL`.
  - `error_code`: contains the error code if an error occurred. Default value is `NULL`.
  - `prompt`: contains the prompt passed to the `ML_RAG` or `ML_GENERATE` routine. Default value is `NULL`. This parameter is populated only if `report_prompt` is set to `true`.
  - `documents`: contains the names of the documents as well as segments used as context by the LLM for response generation. Default value is `NULL`.
  - `request_completed`: set to `true` when a response is the last response message to a request. Default value is `NULL`.
  - `response`: contains the final response from the routine. Default value is `NULL`.

Syntax Examples

- Entering a natural-language query using the `HEATWAVE_CHAT` routine:

```
mysql> CALL sys.HEATWAVE_CHAT("What is Lakehouse?");
```

- Modifying chat parameters using the `@chat_options` variable:
  - Modifying a chat parameter, `tables`, to specify the vector store table to use for context retrieval in the next chat session:

    ```
    mysql> SET @chat_options = '{"tables": [{"table_name": "demo_embeddings", "schema_name": "demo_db"}]}';
    ```

    This example resets the chat session and uses the specified vector store table in the new chat session.
  - Modifying a chat parameter, `tables`, to specify the vector store table to use for context retrieval in the same chat session:

    ```
    mysql> SET @chat_options = JSON_SET(@chat_options,'$.tables', JSON_ARRAY(JSON_OBJECT("table_name", "demo_e
    ```

    This example uses the specified vector store table in the ongoing chat session. It does not reset the chat session.
  - Modifying a chat parameter, `temperature`, without resetting the chat session:

    ```
    mysql> SET @chat_options = json_set(@chat_options, '$.model_options.temperature', 0.5);
    ```

- Viewing the chat parameters and session details:

```
mysql> SELECT JSON_PRETTY(@chat_options);
```

See Also

- Section 5.9, “Starting a Conversational Chat”
- For more information about the output generated by this command, see Section 5.9.2, “Viewing Chat Session Details”.

7.2.7 ML_EMBED_ROW

The `ML_EMBED_ROW` routine uses the specified embedding model to encode the specified text or query into a vector embedding. The routine returns a `VECTOR` that contains a numerical representation of the specified text.

This topic contains the following sections:

- ML_EMBED_ROW Syntax
- Syntax Examples
- See Also

ML_EMBED_ROW Syntax

```
mysql> SELECT sys.ML_EMBED_ROW('Text'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue] ...)
keyvalue:
{
  'model_id', {'ModelID'}
  |'truncate', {true|false}
}
```

Following are `ML_EMBED_ROW` parameters:

- *`Text`*: specifies the text to encode.
- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:
  - `model_id`: specifies the embedding model to use for encoding the text. Default value is `multilingual-e5-small`. Possible values are:
    - `all_minilm_l12_v2`
    - `multilingual-e5-small`

    To view the lists of available embedding models, see In-Database Embedding Model.
  - `truncate`: specifies whether to truncate inputs longer than the maximum token size. Default value is `true`.

Syntax Examples

- Embed an English query using the `all_minilm_l12_v2` embedding model, and store the generated embedding in the `@text_embedding` variable:

  ```
  mysql> SELECT sys.ML_EMBED_ROW("What is artificial intelligence?", JSON_OBJECT("model_id", "all_minilm_l
  ```

  Print the embedding stored in the `@text_embedding` variable:

  ```
  mysql> SELECT @text_embedding;
  ```

  The output, which is a binary representation of the specified text, looks similar to the following:

  ```
  +----------------------------------------------------------------------------------------------------
  | @text_embedding
  +----------------------------------------------------------------------------------------------------
  | 0x97325DBC090E45BDD46399BCC4147C3C73DADFBCE23F41BDB10E1B3D1F5F7D3DE0BEFD3C9A362E3D4258383D643E7ABC4E07
  ```

```
+--------------------------------------------------------------------------------------------------------------
```

To convert the binary representation of this embedding into its string representation, use the `VECTOR_TO_STRING()` function:

```
mysql> SELECT VECTOR_TO_STRING(@text_embedding);
```

The output is similar to the following:

```
+--------------------------------------------------------------------------------------------------------------
| VECTOR_TO_STRING(@text_embedding)
+--------------------------------------------------------------------------------------------------------------
| [-1.35008e-02,-4.81091e-02,-1.87244e-02,1.53858e-02,-2.73258e-02,-4.71801e-02,3.78558e-02,6.18583e-02,3.09
+--------------------------------------------------------------------------------------------------------------
```

The string representation of the embedding consists of a list one or more comma-separated float values, encased in square brackets (`[ ]`). The values are expressed using decimal or scientific notation.

See Also

Section 5.7, "Generating Vector Embeddings"

7.2.8 ML_EMBED_TABLE

The `ML_EMBED_TABLE` routine runs multiple embedding generations in a batch, in parallel.

This topic contains the following sections:

- ML_EMBED_TABLE Syntax
- Syntax Examples
- See Also

To learn about the privileges you need to run this routine, see Section 5.3, "Required Privileges for using GenAI".

ML_EMBED_TABLE Syntax

```
mysql> CALL sys.ML_EMBED_TABLE('InputTableColumn', 'OutputTableColumn'[, options]);

options: JSON_OBJECT(keyvalue[, keyvalue] ...)
keyvalue:
{
  'model_id', {'ModelID'}
  |'truncate', {true|false}
  |'batch_size', BatchSize
  |'details_column', 'ErrorDetailsColumnName'
}
```

Following are `ML_EMBED_TABLE` parameters:

- *`InputTableColumn`*: specifies the names of the input database, table, and column that contains the text to encode. The *`InputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.
  - The specified input table can be an internal or external table.
  - The specified input table must already exist, must not be empty, and must have a primary key.
  - The input column must already exist and must contain `text` or `varchar` values.

    - The input column must not be a part of the primary key and must not have `NULL` values or empty strings.
    - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.

- *`OutputTableColumn`*: specifies the names of the database, table, and column where the generated embeddings are stored. The *`OutputTableColumn`* is specified in the following format: *`DBName`*.*`TableName`*.*`ColumnName`*.

    - The specified output table must be an internal table.
    - If the specified output table already exists, then it must be the same as the input table. And, the specified output column must not already exist in the input table. A new `VECTOR` column is added to the table. External tables are read only. So if input table is an external table, then it cannot be used to store the output.
    - If the specified output table doesn't exist, then a new table is created. The new output table has key columns which contains the same primary key values as the input table and a `VECTOR` column that stores the generated embeddings.
    - There must be no backticks used in the *`DBName`*, *`TableName`*, or *`ColumnName`* and there must be no period used in the *`DBName`* or *`TableName`*.

- `options`: specifies optional parameters as key-value pairs in JSON format. It can include the following parameters:

    - `model_id`: specifies the embedding model to use for encoding the text. Default value is `multilingual-e5-small`. Possible values are:

        - `all_minilm_l12_v2` or `minilm`
        - `multilingual-e5-small`

      To view the lists of available embedding models, see In-Database Embedding Model.

    - `truncate`: specifies whether to truncate inputs longer than the maximum token size. Default value is `true`.
    - `batch_size`: specifies the batch size for the routine. This option is supported for internal tables only. Default value is `1000`. Possible values are integer values between `1` and `1000`.
    - `details_column`: specifies a name for the output table column that is created for adding details of errors encountered for rows that aren't processed successfully by the routine. Ensure that a column by the specified name does not already exist in the table. Default value is `details`.

Syntax Examples

Generate embeddings for text stored in `demo_db.input_table.Input` using the `all_minilm_l12_v2` embedding model, and save the generated embeddings in the output table `demo_db.output_table.Output`:

```
mysql> CALL sys.ML_EMBED_TABLE("demo_db.input_table.Input", "demo_db.output_table.Output", JSON_OBJECT("mod
```

See Also

Generate Vector Embeddings - Run Batch Queries

Chapter 8 Troubleshoot

Table of Contents

The sections in this chapter describe how to troubleshoot MySQL AI errors.

8.1 AutoML Error Messages

Each error message includes an error number, SQLSTATE value, and message string, as described in Error Message Sources and Elements.

- Error number: `ML001016`; SQLSTATE: `HY000`

  Message: Only classification, regression, and forecasting tasks are supported.

  Example: `ERROR HY000: ML001016: Only classification, regression, and forecasting tasks are supported.`

  Check the `task` option in the `ML_TRAIN` call to ensure that it is specified correctly.

- Error number: `ML001031`; SQLSTATE: `HY000`

  Message: Running as a classification task. % classes have less than % samples per class, and cannot be trained on. For a real valued target column, the task parameter in the options JSON should be set to regression.

  Example: `ERROR HY000: ML001031: Running as a classification task. 189 classes have less than 5 samples per class, and cannot be trained on. For a real valued target column, the task parameter in the options JSON should be set to regression.`

  If a classification model is intended, add more samples to the data to increase the minority class count; that is, add more rows with the under-represented target column value. If a classification model was not intended, run `ML_TRAIN` with the regression task option.

- Error number: `ML001051`; SQLSTATE: `HY000`

  Message: One or more rows contain all NaN values. Imputation is not possible on such rows.

  Example: `ERROR HY000: ML001051: One or more rows contain all NaN values. Imputation is not possible on such rows.`

  MySQL does not support NaN values. Replace with NULL.

- Error number: `ML001052`; SQLSTATE: `HY000`

  Message: All columns are dropped. They are constant, mostly unique, or have a lot of missing values!

  Example: `ERROR HY000: ML001052: All columns are dropped. They are constant, mostly unique, or have a lot of missing values!`

  `ML_TRAIN` ignores columns with certain characteristics such as columns missing more than 20% of values and columns containing the same single value. See Section 4.5.1, “Preparing Data”.

- Error number: `ML001053`; SQLSTATE: `HY000`

  Message: Unlabeled samples detected in the training data. (Values in target column can not be NULL).

  Example: `ERROR HY000: ML001053: Unlabeled samples detected in the training data. (Values in target column can not be NULL).`

  Training data must be labeled. See Section 4.5.1, “Preparing Data”.

- Error number: `ML003000`; SQLSTATE: `HY000`

  Message: Number of offloaded datasets has reached the limit!

  Example: `ERROR HY000: ML003000: Number of offloaded datasets has reached the limit!`

- Error number: `ML003011`; SQLSTATE: `HY000`

  Message: Columns of provided data need to match those used for training. Provided - ['%', '%', '%'] vs Trained - ['%', '%'].

  Example: `ERROR HY000: ML003011: Columns of provided data need to match those used for training. Provided - ['petal length', 'petal width', 'sepal length', 'sepal width'] vs Trained - ['petal length', 'sepal length', 'sepal width'].`

  The input data columns do not match the columns of training dataset used to train the model. Compare the input data to the training data to identify the discrepancy.

- Error number: `ML003012`; SQLSTATE: `HY000`

  Message: The table (%) is NULL or has not been loaded.

  Example: `ERROR HY000: ML003012: The table (mlcorpus.iris_train) is NULL or has not been loaded.`

  There is no data in the specified table.

- Error number: `ML003014`; SQLSTATE: `HY000`

  Message: The size of model generated is larger than the maximum allowed.

  Example: `ERROR HY000: ML003014: The size of model generated is larger than the maximum allowed.`

  Models greater than 4 GB in size are not supported.

- Error number: `ML003015`; SQLSTATE: `HY000`

  Message: The input column types do not match the column types of dataset which the model was trained on. ['%', '%'] vs ['%', '%'].

  Example: `ERROR HY000: ML003015: The input column types do not match the column types of dataset which the model was trained on. ['numerical', 'numerical', 'categorical', 'numerical'] vs ['numerical', 'numerical', 'numerical', 'numerical'].`

- Error number: `ML003016`; SQLSTATE: `HY000`

  Message: Missing argument \"row_json\" in input JSON -> dict_keys(['%', '%']).

  Example: `ERROR HY000: ML003016: Missing argument \"row_json\" in input JSON -> dict_keys(['operation', 'user_name', 'table_name', 'schema_name', 'model_handle']).`

- Error number: `ML003017`; SQLSTATE: `HY000`

  Message: The corresponding value of row_json should be a string!

  Example: `ERROR HY000: ML003017: The corresponding value of row_json should be a string!`

- Error number: `ML003018`; SQLSTATE: `HY000`

  Message: The corresponding value of row_json is NOT a valid JSON!

  Example: `ERROR HY000: ML003018: The corresponding value of row_json is NOT a valid JSON!`

- Error number: `ML003019`; SQLSTATE: `HY000`

  Message: Invalid data for the metric (%). Score could not be computed.

  Example: `ERROR HY000: ML003019: Invalid data for the metric (roc_auc). Score could not be computed.`

  The scoring metric is legal and supported, but the data provided is not suitable to calculate such a score. For example: ROC_AUC for multi-class classification. Try a different scoring metric.

- Error number: `ML003020`; SQLSTATE: `HY000`

  Message: Unsupported scoring function (%) for current task (%).

  Example: `ERROR HY000: ML003020: Unsupported scoring function (accuracy) for current task (regression).`

  The scoring metric is legal and supported, but the task provided is not suitable to calculate such a score; for example: Using the `accuracy` metric for a `regression` model.

- Error number: `ML003021`; SQLSTATE: `HY000`

  Message: Cannot train a regression task with a non-numeric target column.

  Example: `ERROR HY000: ML003021: Cannot train a regression task with a non-numeric target column.`

  `ML_TRAIN` was run with the regression task type on a training dataset with a non-numeric target column. Regression models require a numeric target column.

- Error number: `ML003022`; SQLSTATE: `HY000`

  Message: At least 2 target classes are needed for classification task

  Example: `ERROR HY000: ML003022: At least 2 target classes are needed for classification task.`

ML_TRAIN was run with the classification task type on a training dataset where the target column did not have at least two possible values.

- Error number: ML003023; SQLSTATE: 3877 (HY000)

  Message: Unknown option given. Allowed options for training are: ['task', 'model_list', 'exclude_model_list', 'optimization_metric', 'exclude_column_list', 'datetime_index', 'endogenous_variables', 'exogenous_variables', 'positive_class', 'users', 'items', 'user_columns', 'item_columns'].

  Example: ERROR 3877 (HY000): ML003023: Unknown option given. Allowed options for training are: ['task', 'model_list', 'exclude_model_list', 'optimization_metric', 'exclude_column_list', 'datetime_index', 'endogenous_variables', 'exogenous_variables', 'positive_class', 'users', 'items', 'user_columns', 'item_columns'].

  The ML_TRAIN call specified an unknown option.

- Error number: ML003024; SQLSTATE: HY000

  Message: Not enough available memory, unloading any RAPID tables will help to free up memory.

  Example: ERROR HY000: ML003024: Not enough available memory, unloading any RAPID tables will help to free up memory.

  There is not enough memory on the MySQL AI Engine to perform the operation. Try unloading data that was loaded to free up space.

  There might not be enough memory on your system to train the model with large data sets. If this error message appears AutoML, see the system requirements.

- Error number: ML003027; SQLSTATE: 3877 (HY000)

  Message: JSON attribute (item_columns) must be in JSON_ARRAY type.

  Example: ERROR 3877 (HY000): ML003027: JSON attribute (item_columns) must be in JSON_ARRAY type.

  Specify the item_columns JSON attribute as a JSON array.

- Error number: ML003027; SQLSTATE: 3877 (HY000)

  Message: JSON attribute (user_columns) must be in JSON_ARRAY type.

  Example: ERROR 3877 (HY000): ML003027: JSON attribute (user_columns) must be in JSON_ARRAY type.

  Specify the user_columns JSON attribute as a JSON array.

- Error number: ML003039; SQLSTATE: HY000

  Message: Not all user specified columns are present in the input table - missing columns are {%}.

  Example: ERROR HY000: ML003039: Not all user specified columns are present in the input table - missing columns are {C4}.

  The syntax includes a column that is not available.

- Error number: `ML003047`; SQLSTATE: `HY000`

  Message: All columns cannot be excluded. User provided exclude_column_list is ['%', '%'].

  Example: `ERROR HY000: ML003047: All columns cannot be excluded. User provided exclude_column_list is ['C0', 'C1', 'C2', 'C3'].`

  The syntax includes an `exclude_column_list` that attempts to exclude too many columns.

- Error number: `ML003048`; SQLSTATE: `HY000`

  Message: exclude_column_list JSON attribute must be of JSON_ARRAY type.

  Example: `ERROR HY000: ML003048: exclude_column_list JSON attribute must be of JSON_ARRAY type.`

  The syntax includes a malformed `JSON_ARRAY` for the `exclude_column_list`.

- Error number: `ML003048`; SQLSTATE: `HY000`

  Message: include_column_list JSON attribute must be of JSON_ARRAY type.

  Example: `ERROR HY000: ML003048: include_column_list JSON attribute must be of JSON_ARRAY type.`

  The syntax includes a malformed `JSON_ARRAY` for the `include_column_list`.

- Error number: `ML003049`; SQLSTATE: `HY000`

  Message: One or more columns in include_column_list ([%]) does not exist. Existing columns are (['%', '%']).

  Example: `ERROR HY000: ML003049: One or more columns in include_column_list ([C15]) does not exist. Existing columns are (['C0', 'C1', 'C2', 'C3']).`

  The syntax includes an `include_column_list` that expects a column that does not exist.

- Error number: `ML003050`; SQLSTATE: `HY000`

  Message: include_column_list must be a subset of exogenous_variables for forecasting task.

  Example: `ERROR HY000: ML003050: include_column_list must be a subset of exogenous_variables for forecasting task.`

  The syntax for a forecasting task includes an `include_column_list` that expects one or more columns that are not defined by `exogenous_variables`.

- Error number: `ML003052`; SQLSTATE: `HY000`

  Message: Target column provided % is one of the independent variables used to train the model [%, %, %].

  Example: `ERROR HY000: ML003052: Target column provided LSTAT is one of the independent variables used to train the model [RM, RAD, LSTAT].`

  The syntax defines a `target_column_name` that is one of the independent variables used to train the model.

- Error number: `ML003053`; SQLSTATE: `HY000`

Message: datetime_index must be specified by the user for forecasting task and must be a column in the training table.

Example: `ERROR HY000: ML003053: datetime_index must be specified by the user for forecasting task and must be a column in the training table.`

The syntax for a forecasting task must include `datetime_index`, and this must be a column in the training table.

- Error number: `ML003054`; SQLSTATE: `HY000`

  Message: endogenous_variables must be specified by the user for forecasting task and must be column(s) in the training table.

  Example: `ERROR HY000: ML003054: endogenous_variables must be specified by the user for forecasting task and must be column(s) in the training table.`

  The syntax for a forecasting task must include the `endogenous_variables` option, and these must be a column or columns in the training table.

- Error number: `ML003055`; SQLSTATE: `HY000`

  Message: endogenous_variables / exogenous_variables option must be of JSON_ARRAY type.

  Example: `ERROR HY000: ML003055: endogenous_variables / exogenous_variables option must be of JSON_ARRAY type.`

  The syntax for a forecasting task includes `endogenous_variables` or `exogenous_variables` that do not have valid `JSON` format.

- Error number: `ML003056`; SQLSTATE: `HY000`

  Message: exclude_column_list cannot contain any of endogenous or exogenous variables for forecasting task.

  Example: `ERROR HY000: ML003056: exclude_column_list cannot contain any of endogenous or exogenous variables for forecasting task.`

  The syntax for a forecasting task includes `exclude_column_list` that contains columns that are also in `endogenous_variables` or `exogenous_variables`.

- Error number: `ML003057`; SQLSTATE: `HY000`

  Message: endogenous and exogenous variables may not have any common columns for forecasting task.

  Example: `ERROR HY000: ML003057: endogenous and exogenous variables may not have any common columns for forecasting task.`

  The syntax for a forecasting task includes `endogenous_variables` and `exogenous_variables`, and they have one or more columns in common.

- Error number: `ML003058`; SQLSTATE: `HY000`

  Message: Can not train a forecasting task with non-numeric endogenous_variables column(s).

  Example: `ERROR HY000: ML003058: Can not train a forecasting task with non-numeric endogenous_variables column(s).`

  The syntax for a forecasting task includes `endogenous_variables` and some of the columns are not defined as numeric.

- Error number: `ML003059`; SQLSTATE: `HY000`

  Message: User provided list of models ['ThetaForecaster', 'ETSForecaster', 'SARIMAXForecaster', 'ExpSmoothForecaster'] does not include any supported models for the task. Supported models for the given task and table are ['DynFactorForecaster', 'VARMAXForecaster'].

  Example: `ERROR HY000: ML003059: User provided list of models ['ThetaForecaster', 'ETSForecaster', 'SARIMAXForecaster', 'ExpSmoothForecaster'] does not include any supported models for the task. Supported models for the given task and table are ['DynFactorForecaster', 'VARMAXForecaster'].`

  The syntax for a forecasting task includes multivariate `endogenous_variables`, but the provided models only support univariate `endogenous_variables`.

- Error number: `ML003060`; SQLSTATE: `HY000`

  Message:: endogenous_variables may not contain repeated column names ['%1', '%2', '%1'].

  Example: `ERROR HY000: ML003060: endogenous_variables may not contain repeated column names ['wind', 'solar', 'wind'].`

  The syntax for a forecasting task includes `endogenous_variables` with a repeated column.

- Error number: `ML003061`; SQLSTATE: `HY000`

  Message: exogenous_variables may not contain repeated column names ['consumption', 'wind_solar', 'consumption'].

  Example: `ERROR HY000: ML003061: exogenous_variables may not contain repeated column names ['consumption', 'wind_solar', 'consumption'].`

  The syntax for a forecasting task includes `exogenous_variables` with a repeated column.

- Error number: `ML003062`; SQLSTATE: `HY000`

  Message: endogenous_variables argument must not be NULL.

  Example: `ERROR HY000: ML003062: endogenous_variables argument must not be NULL.`

  The syntax for a forecasting task includes `endogenous_variables` with a NULL argument.

- Error number: `ML003063`; SQLSTATE: `HY000`

  Message: exogenous_variables argument must not be NULL when provided by user.

  Example: `ERROR HY000: ML003063: exogenous_variables argument must not be NULL when provided by user.`

  The syntax for a forecasting task includes user provided `exogenous_variables` with a NULL argument.

- Error number: `ML003064`; SQLSTATE: `HY000`

  Message: Cannot exclude all models.

  Example: `ERROR HY000: ML003064: Cannot exclude all models.`

  The syntax for a forecasting task must include at least one model.

- Error number: `ML003065`; SQLSTATE: `HY000`

  Message: Prediction table cannot have overlapping datetime_index with train table when exogenous_variables are used. It can only forecast into future.

  Example: `ERROR HY000: ML003065: Prediction table cannot have overlapping datetime_index with train table when exogenous_variables are used. It can only forecast into future.`

  The syntax for a forecasting task includes `exogenous_variables` and the prediction table contains values in the `datetime_index` column that overlap with values in the `datetime_index` column in the training table.

- Error number: `ML003066`; SQLSTATE: `HY000`

  Message: datetime_index for test table must not have missing dates after the last date in training table. Please ensure test table starts on or before 2034-01-01 00:00:00. Currently, start date in the test table is 2036-01-01 00:00:00.

  Example: `ERROR HY000: ML003066: datetime_index for test table must not have missing dates after the last date in training table. Please ensure test table starts on or before 2034-01-01 00:00:00. Currently, start date in the test table is 2036-01-01 00:00:00.`

  The syntax for a forecasting task includes a prediction table that contains values in the `datetime_index` column that leave a gap to the values in the `datetime_index` column in the training table.

- Error number: `ML003067`; SQLSTATE: `HY000`

  Message: datetime_index for forecasting task must be between year 1678 and 2261.

  Example: `ERROR HY000: ML003067: datetime_index for forecasting task must be between year 1678 and 2261.`

  The syntax for a forecasting task includes values in a `datetime_index` column that are outside the date range from 1678 to 2261.

- Error number: `ML003068`; SQLSTATE: `HY000`

  Message: Last date of datetime_index in the training table 2151-01-01 00:00:00 plus the length of the table 135 must be between year 1678 and 2261.

  Example: `ERROR HY000: ML003068: Last date of datetime_index in the training table 2151-01-01 00:00:00 plus the length of the table 135 must be between year 1678 and 2261.`

  The syntax for a forecasting task includes a prediction table that has too many rows, and the values in the `datetime_index` column would be outside the date range from 1678 to 2261.

- Error number: `ML003070`; SQLSTATE: `3877 (HY000)`

  Message: For recommendation tasks both user and item column names should be provided.

  Example: `ERROR 3877 (HY000): ML003070: For recommendation tasks both user and item column names should be provided.`

- Error number: `ML003071`; SQLSTATE: `HY000`

  Message: contamination must be numeric value greater than 0 and less than 0.5.

  Example: `ERROR HY000: ML003071: contamination must be numeric value greater than 0 and less than 0.5.`

- Error number: `ML003071`; SQLSTATE: `3877 (HY000)`

  Message: item_columns can not contain repeated column names ['C4', 'C4'].

  Example: `ERROR 3877 (HY000): ML003071: item_columns can not contain repeated column names ['C4', 'C4'].`

- Error number: `ML003071`; SQLSTATE: `3877 (HY000)`

  Message: user_columns can not contain repeated column names ['C4', 'C4'].

  Example: `ERROR 3877 (HY000): ML003071: user_columns can not contain repeated column names ['C4', 'C4'].`

- Error number: `ML003072`; SQLSTATE: `HY000`

  Message: Can not use more than one threshold method.

  Example: `ERROR HY000: ML003072: Can not use more than one threshold method.`

- Error number: `ML003072`; SQLSTATE: `3877 (HY000)`

  Message: Target column C3 can not be specified as a user or item column.

  Example: `ERROR 3877 (HY000): ML003072: Target column C3 can not be specified as a user or item column.`

- Error number: `ML003073`; SQLSTATE: `HY000`

  Message: topk must be an integer value between 1 and length of the table, inclusively (1 <= topk <= 20).

  Example: `ERROR HY000: ML003073: topk must be an integer value between 1 and length of the table, inclusively (1 <= topk <= 20).`

- Error number: ML003073; SQLSTATE: 3877 (HY000)

  Message: The users and items columns should be different.

  Example: ERROR 3877 (HY000): ML003073: The users and items columns should be different.

- Error number: ML003074; SQLSTATE: HY000

  Message: threshold must be a numeric value between 0 and 1, inclusively (0 <= threshold <= 1).

  Example: ERROR HY000: ML003074: threshold must be a numeric value between 0 and 1, inclusively (0 <= threshold <= 1).

- Error number: ML003074; SQLSTATE: 3877 (HY000)

  Message: Unsupported ML Operation for recommendation task.

  Example: ERROR 3877 (HY000): ML003074: Unsupported ML Operation for recommendation task.

- Error number: ML003075; SQLSTATE: HY000

  Message: Unknown option given. This scoring metric only allows for these options: ['topk'].

  Example: ERROR HY000: ML003075: Unknown option given. This scoring metric only allows for these options: ['topk'].

- Error number: ML003075; SQLSTATE: 3877 (HY000)

  Message: Unknown option given. Allowed options for recommendations are ['recommend', 'top'].

  Example: ERROR 3877 (HY000): ML003075: Unknown option given. Allowed options for recommendations are ['recommend', 'top'].

- Error number: ML003076; SQLSTATE: HY000

  Message: ML_EXPLAIN, ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE are not supported for anomaly_detection task.

  Example: ERROR HY000: ML003076: ML_EXPLAIN, ML_EXPLAIN_ROW and ML_EXPLAIN_TABLE are not supported for anomaly_detection task.

- Error number: ML003076; SQLSTATE: 3877 (HY000)

  Message: The recommend option should be provided when a value for topk is assigned.

  Example: ERROR 3877 (HY000): ML003076: The recommend option should be provided when a value for topk is assigned.

- Error number: ML003077; SQLSTATE: HY000

  Message: topk must be provided as an option when metric is set as precision_at_k.

  Example: ERROR HY000: ML003077: topk must be provided as an option when metric is set as precision_at_k.

- Error number: ML003077; SQLSTATE: 3877 (HY000)

Message: Unknown recommend value given. Allowed values for recommend are ['ratings', 'items', 'users'].

Example: `ERROR 3877 (HY000): ML003077: Unknown recommend value given. Allowed values for recommend are ['ratings', 'items', 'users'].`

- Error number: `ML003078`; SQLSTATE: `HY000`

  Message: anomaly_detection only allows 0 (normal) and 1 (anomaly) for labels in target column with any metric used, and they have to be integer values.

  Example: `ERROR HY000: ML003078: anomaly_detection only allows 0 (normal) and 1 (anomaly) for labels in target column with any metric used, and they have to be integer values.`

- Error number: `ML003078`; SQLSTATE: `3877 (HY000)`

  Message: Should not provide a value for topk when the recommend option is set to ratings.

  Example: `ERROR 3877 (HY000): ML003078: Should not provide a value for topk when the recommend option is set to ratings.`

- Error number: `ML003079`; SQLSTATE: `3877 (HY000)`

  Message: Provided value for option topk is not a strictly positive integer.

  Example: `ERROR 3877 (HY000): ML003079: Provided value for option topk is not a strictly positive integer.`

- Error number: `ML003080`; SQLSTATE: `3877 (HY000)`

  Message: One or more rows contains NULL or empty values. Please provide inputs without NULL or empty values for recommendation.

  Example: `ERROR 3877 (HY000): ML003080: One or more rows contains NULL or empty values. Please provide inputs without NULL or empty values for recommendation.`

- Error number: `ML003081`; SQLSTATE: `3877 (HY000)`

  Message: Options should be NULL. Options are currently not supported for this task classification.

  Example: `ERROR 3877 (HY000): ML003081: options should be NULL. Options are currently not supported for this task classification.`

- Error number: `ML003082`; SQLSTATE: `3877 (HY000)`

  Message: All supported models are excluded, but at least one model should be included.

  Example: `ERROR 3877 (HY000): ML003082: All supported models are excluded, but at least one model should be included.`

- Error number: `ML003083`; SQLSTATE: `HY000`

  Message: Both user column name ['C3'] and item column name C0 must be provided as string.

  Example: `ERROR HY000: ML003083: Both user column name ['C3'] and item column name C0 must be provided as string.`

- Error number: `ML003105`; SQLSTATE: `3877 (HY000)`

  Message: Cannot recommend users to a user not present in the training table.

  Example: `ERROR: 3877 (HY000): ML003105: Cannot recommend users to a user not present in the training table.`

- Error number: `ML003106`; SQLSTATE: `3877 (HY000)`

  Message: Cannot recommend items to an item not present in the training table.

  Example: `ERROR 3877 (HY000): ML003106: Cannot recommend items to an item not present in the training table.`

- Error number: `ML003107`; SQLSTATE: `3877 (HY000)`

  Message: Users to users recommendation is not supported, please retrain your model.

  Example: `ERROR 3877 (HY000): ML003107: Users to users recommendation is not supported, please retrain your model.`

- Error number: `ML003108`; SQLSTATE: `3877 (HY000)`

  Message: Items to items recommendation is not supported, please retrain your model.

  Example: `ERROR 3877 (HY000): ML003108: Items to items recommendation is not supported, please retrain your model.`

- Error number: `ML003109`; SQLSTATE: `HY000`

  Message: Invalid Model format.

  Example: `HY000: ML003109: Invalid Model format.`

- Error number: `ML003111`; SQLSTATE: `HY000`

  Message: Unknown option given. Allowed options are ['batch_size'].

  Example: `ERROR HY000: ML003111: Unknown option given. Allowed options are ['batch_size'].`

- Error number: `ML003112`; SQLSTATE: `HY000`

  Message: NULL values are not supported for text columns.

  Example: `ERROR HY000: ML003112: NULL values are not supported for text columns.`

- Error number: `ML003114`; SQLSTATE: `HY000`

  Message: Error while parsing text. One of the text columns only contains stop words like the, is, and, a, an, in, has, etc.

  Example: `ERROR HY000: ML003114: Error while parsing text. One of the text columns only contains stop words like the, is, and, a, an, in, has, etc.`

- Error number: `ML003115`; SQLSTATE: `HY000`

  Message: Empty input table after applying threshold.

Example: `ERROR HY000: ML003115: Empty input table after applying threshold.`

- Error number: `ML003116`; SQLSTATE: `HY000`

  Message: The feedback_threshold option can only be set for implicit feedback.

  Example: `ERROR HY000: ML003116: The feedback_threshold option can only be set for implicit feedback.`

- Error number: `ML003117`; SQLSTATE: `HY000`

  Message: The remove_seen option can only be used with the following recommendation ['items', 'users', 'users_to_items', 'items_to_users'].

  Example: `ERROR HY000: ML003117: The remove_seen option can only be used with the following recommendation ['items', 'users', 'users_to_items', 'items_to_users'].`

- Error number: `ML003118`; SQLSTATE: `HY000`

  Message: The remove_seen option must be set to either True or False. Provided *`input`*.

  Example: `ERROR HY000: ML003118: The remove_seen option must be set to either True or False. Provided input.`

- Error number: `ML003119`; SQLSTATE: `HY000`

  Message: The feedback option must either be set to explicit or implicit. Provided *`input`*.

  Example: `ERROR HY000: ML003119: The feedback option must either be set to explicit or implicit. Provided input.`

- Error number: `ML003120`; SQLSTATE: `HY000`

  Message: The input table needs to contain strictly more than one unique item.

  Example: `ERROR HY000: ML003120: The input table needs to contain strictly more than one unique item.`

- Error number: `ML003121`; SQLSTATE: `HY000`

  Message: The input table needs to contain at least one unknown or negative rating.

  Example: `ERROR HY000: ML003121: The input table needs to contain at least one unknown or negative rating.`

- Error number: `ML003122`; SQLSTATE: `HY000`

  Message: The feedback_threshold option must be numeric.

  Example: `ERROR HY000: ML003122: The feedback_threshold option must be numeric.`

- Error number: `ML003123`; SQLSTATE: `HY000`

  Message: User and item columns should contain strings.

  Example: `ERROR HY000: ML003123: User and item columns should contain strings.`

- Error number: `ML003124`; SQLSTATE: `HY000`

  Message: Calculation for precision_at_k metric could not complete because there are no recommended items.

  Example: `ERROR HY000: ML003124: Calculation for precision_at_k metric could not complete because there are no recommended items.`

- Error number: `ML004002`; SQLSTATE: `HY000`

  Message: Output format of onnx model is not supported (output_name={%},output_shape={%},output_type={%}).

  Example: `HY000: ML004002: Output format of onnx model is not supported (output_name={%},output_shape={%},output_type={%}).`

- Error number: `ML004003`; SQLSTATE: `HY000`

  Message: This ONNX model only supports fixed batch size=%.

  Example: `HY000: ML004003: This ONNX model only supports fixed batch size=%.`

- Error number: `ML004005`; SQLSTATE: `HY000`

  Message: The type % in data_types_map is not supported.

  Example: `HY000: ML004005: The type % in data_types_map is not supported.`

- Error number: `ML004006`; SQLSTATE: `HY000`

  Message: ML_SCORE is not supported for an onnx model that does not support batch inference.

  Example: `HY000: ML004006: ML_SCORE is not supported for an onnx model that does not support batch inference.`

- Error number: `ML004007`; SQLSTATE: `HY000`

  Message: ML_EXPLAIN is not supported for an onnx model that does not support batch inference.

  Example: `HY000: ML004007: ML_EXPLAIN is not supported for an onnx model that does not support batch inference.`

- Error number: `ML004008`; SQLSTATE: `HY000`

  Message: onnx model input type=% is not supported! Providing the appropriate types map using 'data_types_map' in model_metadata may resolve the issue.

  Example: `HY000: ML004008: onnx model input type=% is not supported! Providing the appropriate types map using 'data_types_map' in model_metadata may resolve the issue.`

- Error number: `ML004009`; SQLSTATE: `HY000`

  Message: Input format of onnx model is not supported (onnx_input_name={%}, expected_input_shape={%}, expected_input_type={%}, data_shape={%}).

  Example: `HY000: ML004009: Input format of onnx model is not supported (onnx_input_name={%}, expected_input_shape={%}, expected_input_type={%}, data_shape={%}).`

- Error number: `ML004010`; SQLSTATE: `HY000`

  Message: Output being sparse tensor with batch size > 1 is not supported.

  Example: `HY000: ML004010: Output being sparse tensor with batch size > 1 is not supported.`

- Error number: `ML004010`; SQLSTATE: `3877 (HY000)`

  Message: Received data exceeds maximum allowed length 943718400.

  Example: `ERROR 3877 (HY000): ML004010: Received data exceeds maximum allowed length 943718400.`

- Error number: `ML004011`; SQLSTATE: `HY000`

  Message: predictions_name=% is not valid.

  Example: `HY000: ML004011: predictions_name=% is not valid.`

- Error number: `ML004012`; SQLSTATE: `HY000`

  Message: prediction_probabilities_name=% is not valid.

  Example: `HY000: ML004012: prediction_probabilities_name=% is not valid.`

- Error number: `ML004013`; SQLSTATE: `HY000`

  Message: predictions_name should be provided when task=regression and onnx model generates more than one output.

  Example: `HY000: ML004013: predictions_name should be provided when task=regression and onnx model generates more than one output.`

- Error number: `ML004014`; SQLSTATE: `HY000`

  Message: Missing expected JSON key (%)

  Example: `ERROR HY000: ML004014: Missing expected JSON key (schema_name).`

- Error number: `ML004014`; SQLSTATE: `HY000`

  Message: Incorrect labels_map. labels_map should include the key %

  Example: `HY000: ML004014: Incorrect labels_map. labels_map should include the key %`

- Error number: `ML004015`; SQLSTATE: `HY000`

  Message: Expected JSON string type value for key (%)

  Example: `ERROR HY000: ML004015: Expected JSON string type value for key (schema_name).`

- Error number: `ML004015`; SQLSTATE: `HY000`

  Message: When task=classification, if the user does not provide prediction_probabilities_name for the onnx model, ML_EXPLAIN method=% will not be supported.

  Example: `HY000: ML004015: When task=classification, if the user does not provide prediction_probabilities_name for the onnx model, ML_EXPLAIN method=% will not be supported.` % can be "shap", "fast_shap" or "partial_dependence"

- Error number: `ML004016`; SQLSTATE: `HY000`

  Message: Invalid base64-encoded ONNX string.

  Example: `HY000: ML004016: Invalid base64-encoded ONNX string.`

- Error number: `ML004017`; SQLSTATE: `HY000`

  Message: Invalid ONNX model.

  Example: `HY000: ML004017: Invalid ONNX model.`

- Error number: `ML004018`; SQLSTATE: `HY000`

  Message: Parsing JSON arg: Invalid value. failed!

  Example: `ERROR HY000: ML004018: Parsing JSON arg: Invalid value. failed!`

- Error number: `ML004018`; SQLSTATE: `HY000`

  Message: There are issues in running inference session for the onnx model. This might have happened due to inference on inputs with incorrect names, shapes or types.

  Example: `HY000: ML004018: There are issues in running inference session for the onnx model. This might have happened due to inference on inputs with incorrect names, shapes or types.`

- Error number: `ML004019`; SQLSTATE: `HY000`

  Message: Expected JSON object type value for key (%)

  Example: `ERROR HY000: ML004019: Expected JSON object type value for key (JSON root).`

- Error number: `ML004019`; SQLSTATE: `HY000`

  Message: The computed predictions do not have the right format. This might have happened because the provided predictions_name is not correct.

  Example: `HY000: ML004019: The computed predictions do not have the right format. This might have happened because the provided predictions_name is not correct.`

- Error number: `ML004020`; SQLSTATE: `HY000`

  Message: Operation was interrupted by the user.

  Example: `ERROR HY000: ML004020: Operation was interrupted by the user.`

  If a user-initiated interruption, `Ctrl-C`, is detected during the first phase of AutoML model and table load where a MySQL parallel scan is used in the MySQL AI Engine to read data as of MySQL database and send it to the AI engine, error messaging is handled by the MySQL parallel scan function and directed to `ERROR 1317 (70100): Query execution was interrupted.`. The `ERROR 1317 (70100)` message is reported to the client instead of the `ML004020` error message.

- Error number: `ML004020`; SQLSTATE: `HY000`

  Message: The computed prediction probabilities do not have the right format. This might have happened because the provided prediction_probabilities_name is not correct.

  Example: `HY000: ML004020: The computed prediction probabilities do not have the right format. This might have happened because the provided prediction_probabilities_name is not correct.`

- Error number: `ML004021`; SQLSTATE: `HY000`

  Message: The onnx model and dataset do not match. The onnx model's input=% is not a column in the dataset.

  Example: `HY000: ML004021: The onnx model and dataset do not match. The onnx model's input=% is not a column in the dataset.`

- Error number: `ML004022`; SQLSTATE: `HY000`

  Message: The user does not have access privileges to %.

  Example: `ERROR HY000: ML004022: The user does not have access privileges to ml.foo.`

- Error number: `ML004022`; SQLSTATE: `HY000`

  Message: Labels in y_true and y_pred should be of the same type. Got y_true=% and y_pred=YYY. Make sure that the predictions provided by the classifier coincides with the true labels.

  Example: `HY000: ML004022: Labels in y_true and y_pred should be of the same type. Got y_true=% and y_pred=YYY. Make sure that the predictions provided by the classifier coincides with the true labels.`

- Error number: `ML004026`; SQLSTATE: `HY000`

  Message: A column (%) with an unsupported column type (%) detected!

  Example: `ERROR HY000: ML004026: A column (D1) with an unsupported column type (BINARY) detected!`

- Error number: `ML004051`; SQLSTATE: `HY000`

  Message: Invalid operation.

  Example: `ERROR HY000: ML004051: Invalid operation.`

- Error number: `ML004999`; SQLSTATE: `HY000`

  Message: Error during Machine Learning.

  Example: `ERROR HY000: ML004999: Error during Machine Learning.`

- Error number: `ML006006`; SQLSTATE: `45000`

  Message: target_column_name should be NULL or empty.

  Example: `ERROR 45000: ML006006: target_column_name should be NULL or empty.`

- Error number: `ML006017`; SQLSTATE: `45000`

  Message: model_handle already exists in the Model Catalog.

  Example: `45000: ML006017: model_handle already exists in the Model Catalog.`

- Error number: `ML006020`; SQLSTATE: `45000`

  Message: model_metadata should be a JSON object.

  Example: `45000: ML006020: model_metadata should be a JSON object.`

- Error number: `ML006021`; SQLSTATE: `45000`

  Message: contamination has to be passed with anomaly_detection task.

  Example: `ERROR 45000: ML006021: contamination has to be passed with anomaly_detection task.`

- Error number: `ML006022`; SQLSTATE: `45000`

  Message: Unsupported task.

  Example: `ERROR 45000: ML006022: Unsupported task.`

- Error number: `ML006023`; SQLSTATE: `45000`

  Message: "No model object found" will be raised.

  Example: `45000: ML006023: "No model object found" will be raised.`

- Error number: `ML006027`; SQLSTATE: `1644 (45000)`

  Message: Received results exceed `max_allowed_packet`. Please increase it or lower input options value to reduce result size.

  Example: ``ERROR 1644 (45000): ML006027: Received results exceed `max_allowed_packet`. Please increase it or lower input options value to reduce result size.``

- Error number: `ML006029`; SQLSTATE: `45000`

  Message: model_handle is not Ready.

  Example: `45000: ML006029: model_handle is not Ready.`

- Error number: `ML006030`; SQLSTATE: `45000`

Message: onnx_inputs_info must be a json object.

Example: `ERROR 45000: ML006030: onnx_inputs_info must be a json object.`

- Error number: `ML006031`; SQLSTATE: `45000`

  Message: Unsupported format.

  Example: `45000: ML006031: Unsupported format.`

- Error number: `ML006031`; SQLSTATE: `45000`

  Message: onnx_outputs_info must be a json object.

  Example: `ERROR 45000: ML006031: onnx_outputs_info must be a json object.`

- Error number: `ML006032`; SQLSTATE: `45000`

  Message: data_types_map must be a json object.

  Example: `ERROR 45000: ML006032: data_types_map must be a json object.`

- Error number: `ML006033`; SQLSTATE: `45000`

  Message: labels_map must be a json object.

  Example: `ERROR 45000: ML006033: labels_map must be a json object.`

- Error number: `ML006034`; SQLSTATE: `45000`

  Message: onnx_outputs_info must be provided for task=classification.

  Example: `ERROR 45000: ML006034: onnx_outputs_info must be provided for task=classification.`

- Error number: `ML006035`; SQLSTATE: `45000`

  Message: onnx_outputs_info must only be provided for classification and regression tasks.

  Example: `ERROR 45000: ML006035: onnx_outputs_info must only be provided for classification and regression tasks.`

- Error number: `ML006036`; SQLSTATE: `45000`

  Message: % is not a valid key in onnx_inputs_info.

  Example: `ERROR 45000: ML006036: % is not a valid key in onnx_inputs_info.`

- Error number: `ML006037`; SQLSTATE: `45000`

  Message: % is not a valid key in onnx_outputs_info.

  Example: `ERROR 45000: ML006037: % is not a valid key in onnx_outputs_info.`

- Error number: `ML006038`; SQLSTATE: `45000`

  Message: For task=classification, at least one of predictions_name and prediction_probabilities_name must be provided.

Example: `ERROR 45000: ML006038: For task=classification, at least one of predictions_name and prediction_probabilities_name must be provided.`

- Error number: `ML006039`; SQLSTATE: `45000`

  Message: prediction_probabilities_name must only be provided for task=classification.

  Example: `ERROR 45000: ML006039: prediction_probabilities_name must only be provided for task=classification.`

- Error number: `ML006040`; SQLSTATE: `45000`

  Message: predictions_name must not be an empty string.

  Example: `ERROR 45000: ML006040: predictions_name must not be an empty string.`

- Error number: `ML006041`; SQLSTATE: `45000`

  Message: prediction_probabilities_name must not be an empty string.

  Example: `ERROR 45000: ML006041: prediction_probabilities_name must not be an empty string.`

- Error number: `ML006042`; SQLSTATE: `45000`

  Message: labels_map must only be provided for task=classification.

  Example: `ERROR 45000: ML006042: labels_map must only be provided for task=classification.`

- Error number: `ML006043`; SQLSTATE: `45000`

  Message: When labels_map is provided, prediction_probabilities_name must also be provided.

  Example: `ERROR 45000: ML006043: When labels_map is provided, prediction_probabilities_name must also be provided.`

- Error number: `ML006044`; SQLSTATE: `45000`

  Message: When labels_map is provided, predictions_name must not be provided.

  Example: `ERROR 45000: ML006044: When labels_map is provided, predictions_name must not be provided.`

- Error number: `ML006045`; SQLSTATE: `45000`

  Message: ML_SCORE is not supported for a % task.

  Example: `ERROR 45000: ML006045: ML_SCORE is not supported for a % task.`

- Error number: `ML006046`; SQLSTATE: `45000`

  Message: ML_EXPLAIN is not supported for a % task.

  Example: `ERROR 45000: ML006046: ML_EXPLAIN is not supported for a % task.`

- Error number: `ML006047`; SQLSTATE: `45000`

  Message: onnx_inputs_info must only be provided when format='ONNX'.

Example: `ERROR 45000: ML006047: onnx_inputs_info must only be provided when format='ONNX'.`

- Error number: `ML006048`; SQLSTATE: `45000`

  Message: onnx_outputs_info must only be provided when format='ONNX'.

  Example: `ERROR 45000: ML006048: onnx_outputs_info must only be provided when format='ONNX'.`

- Error number: `ML006049`; SQLSTATE: `45000`

  Message: The length of a key provided in onnx_inputs_info should not be greater than 32 characters.

  Example: `ERROR 45000: ML006049: The length of a key provided in onnx_inputs_info should not be greater than 32 characters.`

- Error number: `ML006050`; SQLSTATE: `45000`

  Message: The length of a key provided in onnx_outputs_info should not be greater than 32 characters.

  Example: `ERROR 45000: ML006050: The length of a key provided in onnx_outputs_info should not be greater than 32 characters.`

- Error number: `ML006051`; SQLSTATE: `45000`

  Message: Invalid ONNX model.

  Example: `ERROR 45000: ML006051: Invalid ONNX model.`

- Error number: `ML006052`; SQLSTATE: `45000`

  Message: Input table is empty. Please provide a table with at least one row.

  Example: `ERROR 45000: ML006052: Input table is empty. Please provide a table with at least one row.`

- Error number: `ML006053`; SQLSTATE: `45000`

  Message: Insufficient access rights. Grant user with correct privileges (SELECT, DROP, CREATE, INSERT, ALTER) on input schema.

  Example: `ERROR 45000: ML006053: Insufficient access rights. Grant user with correct privileges (SELECT, DROP, CREATE, INSERT, ALTER) on input schema.`

- Error number: `ML006054`; SQLSTATE: `45000`

  Message: input table already contains a column named `_4aad19ca6e_pk_id`. Please provide an input table without such column.

  Example: ``ERROR 45000: ML006054: Input table already contains a column named `_4aad19ca6e_pk_id`. Please provide an input table without such column.``

- Error number: `ML006055`; SQLSTATE: `45000`

  Message: Options must be a JSON_OBJECT.

  Example: `ERROR 45000: ML006055: Options must be a JSON_OBJECT.`

- Error number: `ML006056`; SQLSTATE: `45000`

  Message: batch_size must be an integer between 1 and %.

  Example: `ERROR 45000: ML006056: batch_size must be an integer between 1 and %.`

- Error number: `ML006070`; SQLSTATE: `45000`

  Message: model_list is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006070: model_list is currently not supported for anomaly_detection.`

- Error number: `ML006071`; SQLSTATE: `45000`

  Message: exclude_model_list is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006071: exclude_model_list is currently not supported for anomaly_detection.`

- Error number: `ML006072`; SQLSTATE: `45000`

  Message: optimization_metric is currently not supported for anomaly_detection.

  Example: `ERROR 45000: ML006072: optimization_metric is currently not supported for anomaly_detection.`

8.2 GenAI Issues

This section describes some commonly encountered issues and errors for GenAI and their workarounds.

- *Issue:* When you try to verify whether the vector embeddings were correctly loaded, if you see a message which indicates that the vector embeddings or table did not load in MySQL AI, then it could be due one of the following reasons:

  - The task that loads the vector embeddings into the vector store table might still be running.

    Workaround: Check the task status by using the query that was printed by the `VECTOR_STORE_LOAD` routine:

    ```
    SELECT * from mysql_task_management.task_status where id = TaskID;
    ```

    Or, to see the log messages, check the task logs table:

    ```
    SELECT * from mysql_task_management.task_log where task_id = TaskID;
    ```

    Replace *`TaskID`* with the ID for the task which was printed by the `VECTOR_STORE_LOAD` routine.

  - The folder you are trying to load might contain unsupported format files or the file that you are trying to load might be of an unsupported format.

    Workaround: The supported file formats are: PDF, TXT, PPT, HTML, and DOC.

    If you find unsupported format files, then try one of the following:

    - Delete the files with unsupported formats from the folder, and run the `VECTOR_STORE_LOAD` command again to load the vector embeddings into the vector store table again.

    - Move the files with supported formats to another folder, create a new PAR and run the `VECTOR_STORE_LOAD` command with the new PAR to load the vector embeddings into the vector store table again.

- *Issue:* the `VECTOR_STORE_LOAD` command fails unexpectedly.

  Workaround: Ensure that you use the `--sqlc` flag when you connect to your database system:

  ```
  mysqlsh -uAdmin -pPassword -hPrivateIP --sqlc
  ```

  Replace the following:

  - *`Admin`*: the admin name.
  - *`Password`*: the database system password.
  - *`PrivateIP`*: the private IP address of the database system.