Oracle® Machine Learning for SQL

User's Guide

26ai
F47583-12
April 2026

ORACLE

Oracle Machine Learning for SQL User's Guide, 26ai

F47583-12

Primary Author: Sarika Surampudi

Contributors: Mark Hornick, Boriana Milanova

Contents

3 Prepare the Data

4 Create a Model

5 Scoring and Deployment

8 Administrative Tasks for Oracle Machine Learning for SQL

A Oracle Machine Learning for SQL Examples

Index

List of Tables

Preface

This guide explains how to use the programmatic interfaces to Oracle Machine Learning for SQL (OML4SQL), previously known as Oracle Data Mining. This guide also describes how to use features of Oracle AI Database to administer OML4SQL, and presents the tools and procedures for implementing the concepts that are presented in *Oracle Machine Learning for SQL Concepts* .

This preface contains these topics:

Technology Rebrand

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

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

Audience

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

Related Documentation

The following manuals document Oracle Machine Learning for SQL:

- *Oracle Machine Learning for SQL Concepts*
- *Oracle Machine Learning for SQL User's Guide* (this guide)
- *Oracle Machine Learning for SQL API Guide*

> **Note**
>
> This publication combines key passages from the other two Oracle Machine Learning for SQL manuals with related reference documentation in *Oracle Database PL/SQL Packages and Types Reference*, *Oracle Database SQL Language Reference*, and *Oracle Database Reference*.

- *Oracle Database PL/SQL Packages and Types Reference* (PL/SQL packages)
  - `DBMS_DATA_MINING`
  - `DBMS_DATA_MINING_TRANSFORM`
  - `DBMS_PREDICTIVE_ANALYTICS`
- *Oracle Database Reference* (data dictionary views for `ALL_`, `USER_`, and `DBA_`)
  - `ALL_MINING_MODELS`
  - `ALL_MINING_MODEL_ATTRIBUTES`
  - `ALL_MINING_MODEL_SETTINGS`
- *Oracle Database SQL Language Reference* (OML4SQL functions)
  - `CLUSTER_DETAILS`, `CLUSTER_DISTANCE`, `CLUSTER_ID`, `CLUSTER_PROBABILITY`, `CLUSTER_SET`
  - `FEATURE_DETAILS`, `FEATURE_ID`, `FEATURE_SET`, `FEATURE_VALUE`
  - `PREDICTION`, `PREDICTION_BOUNDS`, `PREDICTION_COST`, `PREDICTION_DETAILS`, `PREDICTION_PROBABILITY`, `PREDICTION_SET`
- Oracle Machine Learning for SQL Resources on the Oracle Technology Network
- Application Development and Database Administration Documentation

Oracle Machine Learning for SQL Resources on the Oracle Technology Network

The Oracle Machine Learning for SQL page on the Oracle Technology Network (OTN) provides a wealth of information, including white papers, demonstrations, blogs, discussion forums, and Oracle By Example tutorials.

You can download Oracle Data Miner, the graphical user interface to Oracle Machine Learning for SQL, from this site:

`Oracle Data Miner`

Application Development and Database Administration Documentation

For documentation to assist you in developing database applications and in administering Oracle AI Database, refer to the following:

- *Oracle Database Concepts*

- *Oracle Database Administrator's Guide*
- *Oracle Database Development Guide*

Conventions

The following text conventions are used in this document:

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

Changes in This Release for Oracle Machine Learning for SQL User's Guide

Describes changes in *Oracle Machine Learning for SQL User's Guide* for Oracle AI Database 26ai.

New Features

- Computer Vision (CV) models in your database: Integrate CV models in your database and generate embeddings for images. See Image Transformer ONNX Format Models for more details.
- Vector Data Support: Oracle Machine Learning supports vector data type for clustering, classification, regression, anomaly detection, and feature extraction. See Vector Data Type to learn more.
- Oracle Machine Learning for SQL supports ONNX format models with the integration of ONNX Runtime. To learn more, see Integration of ONNX Runtime.
- `BOOLEAN` data type is supported. For more information, see Convert Column Data Types, Numericals, Categoricals, and Unstructured Text, and Target Attribute.

Enhancements

Oracle AI Database supports in-memory sharing of external initializers for ONNX models, reducing memory usage and improving scalability for concurrent inference workloads. The database loads large initializers once into global memory and shares them across sessions. This feature applies to ONNX models imported with external initializers and supports monitoring and control through dictionary views. See External Initializers and In-Memory Sharing and Enable and Use External Initializers for more details.

Model Views

- Model detail views for ONNX models are introduced. The `DM$VX` and `DM$VM` model detail views are added. See Model Detail Views for ONNX Models
- Model view for Exponential Smoothing is enhanced. See Model Detail Views for Exponential Smoothing.

Other Changes

The following is an additional change in *Oracle Machine Learning for SQL User's Guide* for 23ai:

Throughout the document, short descriptions are updated and minor edits are made for better readability.

1

Oracle Machine Learning With SQL

Learn how to solve business problems using the Oracle Machine Learning for SQL application programming interface (API).

- Highlights of the Oracle Machine Learning for SQL API
  Learn about the advantages of OML4SQL application programming interface (API).
- Example: Predicting Likely Candidates for a Sales Promotion
  This example shows `PREDICTION` query to target customers in Brazil for a special promotion that offers coupons and an affinity card.
- Example: Analyzing Preferred Customers
  The examples in this section reveal information about customers who use affinity cards or are likely to use affinity cards.
- Example: Segmenting Customer Data
  The examples in this section use an Expectation Maximization clustering model to segment the customer data based on common characteristics.
- Example : Comparison of Texts Using an ESA Model
  The examples shows the `FEATURE_COMPARE` function comparing texts for semantic relatedness (similarity) using the Explicit Semantic Analysis (ESA) prebuilt Wikipedia-based model, which extracts topics and compares text.
- Example: Using Vector Data for Dimensionality Reduction and Clustering
  The example demonstrates how to use vector data for dimensionality reduction and clustering, using Principal Component Analysis (PCA) and *k*-Means.

1.1 Highlights of the Oracle Machine Learning for SQL API

Learn about the advantages of OML4SQL application programming interface (API).

Machine learning is a valuable technology in many application domains. It has become increasingly indispensable in the private sector as a tool for optimizing operations and maintaining a competitive edge. Machine learning also has critical applications in the public sector and in scientific research. However, the complexities of machine learning application development and the complexities inherent in managing and securing large stores of data can limit the adoption of machine learning technology.

OML4SQL is uniquely suited to addressing these challenges. The machine learning engine is implemented in the database kernel, and the robust administrative features of Oracle AI Database are available for managing and securing the data. While supporting a full range of machine learning algorithms and procedures, the API also has features that simplify the development of machine learning applications.

The OML4SQL API consists of extensions to Oracle SQL, the native language of the database. The API offers the following advantages:

- Scoring in the context of SQL queries. Scoring can be performed dynamically or by applying machine learning models.
- Automatic Data Preparation (ADP) and embedded transformations.

- Model transparency. Algorithm-specific queries return details about the attributes that were used to create the model.
- Scoring transparency. Details about the prediction, clustering, or feature extraction operation can be returned with the score.
- Simple routines for predictive analytics.
- A workflow-based graphical user interface (GUI) within Oracle SQL Developer. You can download SQL Developer free of charge from the following site:

  `Oracle Data Miner`

> **Note**
>
> The examples in this publication are taken from the OML4SQL examples that are available on GitHub. For information on the examples, see About the OML4SQL Examples.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

1.2 Example: Predicting Likely Candidates for a Sales Promotion

This example shows `PREDICTION` query to target customers in Brazil for a special promotion that offers coupons and an affinity card.

The query uses data on marital status, education, and income to predict the customers who are most likely to take advantage of the incentives. The query applies a Decision Tree model called dt_sh_clas_sample to score the customer data. The model is created by the `oml4sql-classification-decision-tree.sql` example.

Example 1-1 Predict Best Candidates for an Affinity Card

```
SELECT cust_id
  FROM mining_data_apply_v
  WHERE
      PREDICTION(dt_sh_clas_sample
                  USING cust_marital_status, education, cust_income_level ) = 1
  AND country_name IN 'Brazil';
```

The output is as follows:

```
CUST_ID
----------
    100404
    100607
    101113
```

The same query, but with a bias to favor false positives over false negatives, is shown here.

```
SELECT cust_id
  FROM mining_data_apply_v
  WHERE
```

```
      PREDICTION(dt_sh_clas_sample COST MODEL
                  USING cust_marital_status, education, cust_income_level ) = 1
  AND country_name IN 'Brazil';
```

The output is as follows:

```
CUST_ID
----------
    100139
    100163
    100275
    100404
    100607
    101113
    101170
    101463
```

The `COST MODEL` keywords cause the cost matrix associated with the model to be used in making the prediction. The cost matrix, stored in a table called `dt_sh_sample_costs`, specifies that a false negative is eight times more costly than a false positive. Overlooking a likely candidate for the promotion is far more costly than including an unlikely candidate.

```
SELECT * FROM dt_sh_sample_cost;
```

The output is as follows:

```
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ----------------------- ----------
                  0                       0          0
                  0                       1          1
                  1                       0          8
                  1                       1          0
```

1.3 Example: Analyzing Preferred Customers

The examples in this section reveal information about customers who use affinity cards or are likely to use affinity cards.

Example 1-2 Find Demographic Information About Preferred Customers

This query returns the gender, age, and length of residence of typical affinity card holders. The anomaly detection model, SVMO_SH_Clas_sample, returns `1` for typical cases and `0` for anomalies. The demographics are predicted for typical customers only; outliers are not included in the sample. The model is created by the `oml4sql-anomaly-detection-1class-svm.sql` example.

```
SELECT cust_gender, round(avg(age)) age,
       round(avg(yrs_residence)) yrs_residence,
       count(*) cnt
FROM mining_data_one_class_v
WHERE PREDICTION(SVMO_SH_Clas_sample using *) = 1
GROUP BY cust_gender
ORDER BY cust_gender;
```

The output is as follows:

```
CUST_GENDER          AGE YRS_RESIDENCE        CNT
------------ ---------- ------------- ----------
F                     40             4         36
M                     45             5        304
```

Example 1-3 Dynamically Identify Customers Who Resemble Preferred Customers

This query identifies customers who do not currently have an affinity card, but who share many of the characteristics of affinity card holders. The `PREDICTION` and `PREDICTION_PROBABILITY` functions use an `OVER` clause instead of a predefined model to classify the customers. The predictions and probabilities are computed dynamically.

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

The output is as follows:

```
  CUST_ID PRED_PROB
---------- ---------
    102434       .96
    102365       .96
    102330       .96
    101733       .95
    102615       .94
    102686       .94
    102749       .93
.
.
.
.
    102580       .52
    102269       .52
    102533       .51
    101604       .51
    101656       .51

226 rows selected.
```

Example 1-4 Predict the Likelihood that a New Customer Becomes a Preferred Customer

This query computes the probability of a first-time customer becoming a preferred customer (an affinity card holder). This query can be run in real time at the point of sale.

The new customer is a 44-year-old American executive who has a bachelors degree and earns more than $300,000/year. He is married, lives in a household of 3, and has lived in the same

residence for the past 6 years. The probability of this customer becoming a typical affinity card holder is only 5.8%.

```
SELECT PREDICTION_PROBABILITY(SVMO_SH_Clas_sample, 1 USING
                              44 AS age,
                              6 AS yrs_residence,
                              'Bach.' AS education,
                              'Married' AS cust_marital_status,
                              'Exec.' AS occupation,
                              'United States of America' AS country_name,
                              'M' AS cust_gender,
                              'L: 300,000 and above' AS cust_income_level,
                              '3' AS houshold_size
                              ) prob_typical
FROM DUAL;
```

The output is as follows:

```
PROB_TYPICAL
------------
  5.8
```

Example 1-5 Use Predictive Analytics to Find Top Predictors

The `DBMS_PREDICTIVE_ANALYTICS` PL/SQL package contains routines that perform simple machine learning operations without a predefined model. In this example, the `EXPLAIN` routine computes the top predictors for affinity card ownership. The procedure does not create a model that can be stored in the database for further exploration. Automatic Data Preparation is also performed behind the scenes. The results show that household size, marital status, and age are the top three predictors.

```
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'mining_data_test_v',
        explain_column_name  => 'affinity_card',
        result_table_name    => 'cust_explain_result');
END;
/

SELECT * FROM cust_explain_result
  WHERE rank < 4;
```

The output is as follows:

```
ATTRIBUTE_NAME           ATTRIBUTE_SUBNAME    EXPLANATORY_VALUE       RANK
------------------------ -------------------- ----------------- ----------
HOUSEHOLD_SIZE                                      .209628541          1
CUST_MARITAL_STATUS                                 .199794636          2
AGE                                                 .111683067          3
```

Another way to arrive at top predictors for affinity ownership is by using attribute importance mining function. Create a model with the Minimum Description Length algorithm. Define

mining_function as ATTRIBUTE_IMPORTANCE. You can then query the DM$VA model detail view to get the top three predictors.

```
BEGIN DBMS_DATA_MINING.DROP_MODEL('AI_EXPLAIN_OUTPUT');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME') := 'ALGO_AI_MDL';
    V_setlst('PREP_AUTO') := 'ON';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME => 'AI_EXPLAIN_OUTPUT',
        MINING_FUNCTION => 'ATTRIBUTE_IMPORTANCE',
        DATA_QUERY => 'select * from mining_data_test_v',
        SET_LIST => v_setlst,
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        TARGET_COLUMN_NAME => 'AFFINITY_CARD');
END;


Find the top 3 predictors from the DM$VA model detail view:
SELECT ATTRIBUTE_NAME, ATTRIBUTE_IMPORTANCE_VALUE, ATTRIBUTE_RANK FROM
DM$VAAI_EXPLAIN_OUTPUT;
```

The output is as follows:

```
ATTRIBUTE_NAME            ATTRIBUTE_IMPORTANCE_VALUE   ATTRIBUTE_RANK
HOUSEHOLD_SIZE            0.16154338717879052                         1
CUST_MARITAL_STATUS       0.1561477632217005                          2
AGE                       0.08440594628406521                         3
```

1.4 Example: Segmenting Customer Data

The examples in this section use an Expectation Maximization clustering model to segment the customer data based on common characteristics.

Example 1-6 Compute Customer Segments

This query computes natural groupings of customers and returns the number of customers in each group. The em_sh_clus_sample model is created by the oml4sql-clustering-expectation-maximization.sql example.

```
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
  FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;
```

The output is as follows:

```
      CLUS        CNT
---------- ----------
```

```
         9        311
         3        294
         7        215
        12        201
        17        123
        16        114
        14         86
        19         64
        15         56
        18         36
```

Example 1-7 Find the Customers Who Are Most Likely To Be in the Largest Segment

The query in Example 1-6 shows that segment 9 has the most members. The following query lists the five customers who are most likely to be in segment 9.

```
SELECT cust_id
FROM (SELECT cust_id, RANK() over (ORDER BY prob DESC, cust_id) rnk_clus2
  FROM (SELECT cust_id,
          ROUND(CLUSTER_PROBABILITY(em_sh_clus_sample, 9 USING *),3) prob
          FROM mining_data_apply_v))
WHERE rnk_clus2 <= 5
ORDER BY rnk_clus2;
```

The output is as follows:

```
   CUST_ID
----------
    100002
    100012
    100016
    100019
    100021
```

Example 1-8 Find Key Characteristics of the Most Representative Customer in the Largest Cluster

The query in Example 1-7 lists customer 100002 first in the list of likely customers for segment 9. The following query returns the five characteristics that are most significant in determining the assignment of customer 100002 to segments with probability > 20% (only segment 9 for this customer).

```
SELECT S.cluster_id, probability prob,
       CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 using T.*) det
 FROM
  (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
    FROM mining_data_apply_v v
    WHERE cust_id = 100002) T,
 TABLE(T.pset) S
 ORDER BY 2 desc;
```

The output is as follows:

```
CLUSTER_ID    PROB DET
---------- -------
--------------------------------------------------------------------------------
```

```
--
         9  1.0000 <Details algorithm="Expectation Maximization" cluster="9">
                   <Attribute name="YRS_RESIDENCE" actualValue="4" weight="1"
rank="1"/>
                   <Attribute name="EDUCATION" actualValue="Bach." weight="0"
rank="2"/>
                   <Attribute name="AFFINITY_CARD" actualValue="0" weight="0"
rank="3"/>
                   <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight="0" rank="4"/>
                   <Attribute name="Y_BOX_GAMES" actualValue="0" weight="0"
rank="5"/>
                   </Details>
```

1.5 Example : Comparison of Texts Using an ESA Model

The examples shows the `FEATURE_COMPARE` function comparing texts for semantic relatedness (similarity) using the Explicit Semantic Analysis (ESA) prebuilt Wikipedia-based model, which extracts topics and compares text.

The examples shows an ESA model built against a prebuilt Wiki data set rendering over 200,000 features. The documents are analyzed as text and the document titles are given as the feature IDs. In the first example, the pair of sentence scores higher because Nick Price is a golfer born in South Africa.

Similar Texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA tour
golfers from South Africa' text AND USING 'Nick Price won the 2002 Mastercard
Colonial Open' text) similarity FROM DUAL;
```

The output is as follows:

```
SIMILARITY
----------
      .110
```

The output metric shows distance calculation. Therefore, smaller number represent more similar texts. So, `1` minus the distance in the queries result in similarity.

Dissimilar Texts

```
SELECT 1-FEATURE_COMPARE(esa_wiki_mod USING 'There are several PGA tour
golfers from South Africa' text AND USING 'John Elway played quarterback for
the Denver Broncos' text) similarity FROM DUAL;
```

The output is as follows:

```
SIMILARITY
----------
```

```
.004
```

1.6 Example: Using Vector Data for Dimensionality Reduction and Clustering

The example demonstrates how to use vector data for dimensionality reduction and clustering, using Principal Component Analysis (PCA) and *k*-Means.

1. Assume that there is a data set called `datavec` containing one `ID` column and a vector column with 100 dimensions.

```
Name                  Null?    Type
--------------------- -------- ----------------------------
ID                             NUMBER
PROD_DATA                      VECTOR(100, FLOAT32, DENSE)
```

2. Build a PCA feature extraction model. The following step creates a model that uses PCA scoring to reduce dimensionality.

```
DECLARE
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  v_setlst('ALGO_NAME')         := 'ALGO_SINGULAR_VALUE_DECOMP';
  v_setlst('SVDS_SCORING_MODE') := 'SVDS_SCORING_PCA';

  DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'pca_model',
        MINING_FUNCTION     => 'FEATURE_EXTRACTION',
        DATA_QUERY          => 'SELECT * FROM DATAVEC',
        CASE_ID_COLUMN_NAME => 'id',
        SET_LIST            => v_setlst);
END;
/
```

3. Transform PCA results into a vector table `pca_data` with reduced dimensions by using the `VECTOR_EMBEDDING()` operator.

```
CREATE table pca_data as SELECT id, VECTOR_EMBEDDING(pca_model using *)
embedding FROM datavec;
```

4. The new `pca_data` contains one ID column and one vector with 10 dimensions based on the data characteristics.

```
DESC pca_data;
Name             Null?    Type
---------------- -------- ----------------------------
ID                         NUMBER
EMBEDDING                 VECTOR(10, FLOAT64, DENSE)
```

5. Build a *k*-Means clustering model on `pca_data`, leveraging its reduced dimensions.

```
DECLARE
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  v_setlst('ALGO_NAME')         := 'ALGO_KMEANS';
  v_setlst('KMNS_DETAILS')      := 'KMNS_DETAILS_ALL';
  v_setlst('CLUS_NUM_CLUSTERS') := '2';

  DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'km_model',
        MINING_FUNCTION     => 'CLUSTERING',
        DATA_QUERY          => 'SELECT * FROM PCA_DATA',
        CASE_ID_COLUMN_NAME => 'id',
        SET_LIST            => v_setlst);
END;
/
```

6. Check the data dictionary settings.

```
SELECT model_name, attribute_name, data_type, target, vector_info
FROM   USER_MINING_MODEL_ATTRIBUTES
WHERE  model_name='KM_MODEL' ORDER BY attribute_name;

MODEL_NAME ATTRIBUTE_NAME  DATA_TYPE            TAR VECTOR_INFO
---------- --------------- ----------------------- ---
--------------------
KM_MODEL   EMBEDDING       VECTOR                  NO  VECTOR(10,FLOAT64)
```

7. You can check the model detail views for `KM_MODEL` model.

```
SELECT model_name, view_name, view_type
FROM USER_MINING_MODEL_VIEWS
WHERE model_name='KM_MODEL' ORDER BY view_name;

MODEL_NAME VIEW_NAME         VIEW_TYPE
---------- --------------- ----------------------------------------
KM_MODEL   DM$VAKM_MODEL   Clustering Attribute Statistics
KM_MODEL   DM$VCKM_MODEL   k-Means Scoring Centroids
KM_MODEL   DM$VDKM_MODEL   Clustering Description
KM_MODEL   DM$VGKM_MODEL   Global Name-Value Pairs
KM_MODEL   DM$VHKM_MODEL   Clustering Histograms
KM_MODEL   DM$VNKM_MODEL   Normalization and Missing Value Handling
KM_MODEL   DM$VRKM_MODEL   Clustering Rules
KM_MODEL   DM$VSKM_MODEL   Computed Settings
KM_MODEL   DM$VWKM_MODEL   Model Build Alerts
```

8. You can also view each vector dimension as a predictor from the model details.

```
SELECT * FROM(SELECT cluster_id, attribute_name, attribute_subname,
       mean, variance, mode_value
FROM DM$VAKM_MODEL ORDER BY cluster_id, attribute_name,attribute_subname)

CLUSTER_ID ATTRIBUTE_NAME  ATTRIBUTE_SUBNAME    MEAN
VARIANCE       MODE_VALUE
```

```
---------- -------------- ------------------- -------------
------------ --------------------
         1 EMBEDDING          DM$$VEC1             28.9538       3.4382
         2 EMBEDDING          DM$$VEC1             27.9580       5.5661
         3 EMBEDDING          DM$$VEC1             29.9495       2.1698
```

9. Use scoring operators `CLUSTER_ID` and `CLUSTER_PROBABILITY` to find cluster assignments and probabilities for each record in `pca_data`.

```
SELECT id, cluster_id(km_model using *) cluster_id,
cluster_probability(km_model using *)probability FROM pca_data ORDER BY id;

ID         CLUSTER_ID      PROBABILITY
----------      ----------      -----------
1               1               .617
2               2               .584
3               1               .579
4               1               .605
5               1               .621
6               1               .642
7               2               .598
8               2               .614
9               2               .650
10              2               .618
```

2

About the Oracle Machine Learning for SQL API

Overview of the OML4SQL application programming interface (API) components.

- About Oracle Machine Learning Models
  Machine learning models are database schema objects that perform machine learning techniques.
- Oracle Machine Learning Data Dictionary Views
  Lists Oracle Machine Learning data dictionary views.
- Oracle Machine Learning Modeling, Transformations, and Convenience Functions
  You can access PL/SQL interface to perform data modeling, transformations, and predictive analytics.
- Oracle Machine Learning for SQL Scoring Functions
  Use OML4SQL functions score data. Functions can apply a machine learning model schema object to data or dynamically mine it with an analytic clause. SQL functions exist for all OML4SQL scoring algorithms.
- Oracle Machine Learning for SQL Statistical Functions
  Various SQL statistical functions are available in Oracle AI Database to explore and analyze data.

2.1 About Oracle Machine Learning Models

Machine learning models are database schema objects that perform machine learning techniques.

As with all schema objects, access to machine learning models is controlled by database privileges. Models can be exported and imported. They support comments and they can be tracked in the Oracle AI Database auditing system.

Machine learning models are created by the `CREATE_MODEL2` or the `CREATE_MODEL` procedures in the `DBMS_DATA_MINING` PL/SQL package. Models are created for a specific machine learning technique, and they use a specific algorithm to perform that function. **Machine learning technique** is a term that refers to a class of machine learning problems to be solved. Examples of machine learning techniques are: regression, classification, attribute importance, clustering, anomaly detection, and feature selection. OML4SQL supports one or more algorithms for each machine learning technique.

Along with the machine learning technique, in the `CREATE_MODEL2` procedure, you can specify an algorithm and other characteristics of a model. In `CREATE_MODEL` procedure you can specify a settings table to specify an algorithm and other characteristics of a model. Some settings are general, some are specific to a machine learning technique, and some are specific to an algorithm.

> **Note**
>
> Most types of machine learning models can be used to score data. However, it is possible to score data without applying a model. Dynamic scoring and predictive analytics return scoring results without a user-supplied model. They create and apply transient models that are not visible to you.

Related Topics

- Create a Model
  Explains how to create Oracle Machine Learning for SQL models and to query model details.
- Administrative Tasks for Oracle Machine Learning for SQL
  Explains how to perform administrative tasks related to Oracle Machine Learning for SQL.
- Dynamic Scoring
  You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.
- DBMS_PREDICTIVE_ANALYTICS
  The `DBMS_PREDICTIVE_ANALYTICS` package contains routines that perform an automated form of machine learning known as predictive analytics. With predictive analytics, you do not need to be aware of model building or scoring. All machine learning activities are handled internally by the procedure.

2.2 Oracle Machine Learning Data Dictionary Views

Lists Oracle Machine Learning data dictionary views.

The data dictionary views for Oracle Machine Learning are listed in the following table. A database administrator (DBA) and USER versions of the views are also available.

Table 2-1 Data Dictionary Views for Oracle Machine Learning

View Name	Description
ALL_MINING_MODELS	Provides information about all accessible machine learning models
ALL_MINING_MODEL_ATTRIBUTES	Provides information about the attributes of all accessible machine learning models
ALL_MINING_MODEL_PARTITIONS	Provides information about the partitions of all accessible partitioned machine learning models
ALL_MINING_MODEL_SETTINGS	Provides information about the configuration settings for all accessible machine learning models
ALL_MINING_MODEL_VIEWS	Provides information about the model views for all accessible machine learning models
ALL_MINING_MODEL_XFORMS	Provides the user-specified transformations embedded in all accessible machine learning models.

- ALL_MINING_MODELS
  Describes an example of `ALL_MINING_MODELS` and shows a sample query.
- ALL_MINING_MODEL_ATTRIBUTES
  Describes an example of `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query.

- ALL_MINING_MODEL_PARTITIONS
  Describes an example of ALL_MINING_MODEL_PARTITIONS and shows a sample query.
- ALL_MINING_MODEL_SETTINGS
  Describes an example of ALL_MINING_MODEL_SETTINGS and shows a sample query.
- ALL_MINING_MODEL_VIEWS
  Describes an example of ALL_MINING_MODEL_VIEWS and shows a sample query.
- ALL_MINING_MODEL_XFORMS
  Describes an example of ALL_MINING_MODEL_XFORMS and provides a sample query.

2.2.1 ALL_MINING_MODELS

Describes an example of ALL_MINING_MODELS and shows a sample query.

The following example describes ALL_MINING_MODELS and shows a sample query.

Example 2-1 ALL_MINING_MODELS

```
 describe ALL_MINING_MODELS
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 MINING_FUNCTION                                    VARCHAR2(30)
 ALGORITHM                                          VARCHAR2(30)
 CREATION_DATE                             NOT NULL DATE
 BUILD_DURATION                                     NUMBER
 MODEL_SIZE                                         NUMBER
 BUILD_SOURCE                           CLOB
 PARTITIONED                                        VARCHAR2(3)
 COMMENTS                                           VARCHAR2(4000)
```

The following query returns the models accessible to you that use the Support Vector Machine algorithm.

```
SELECT mining_function, model_name
    FROM all_mining_models
    WHERE algorithm = 'SUPPORT_VECTOR_MACHINES'
    ORDER BY mining_function, model_name;



MINING_FUNCTION
MODEL_NAME
------------------------
--------------------
CLASSIFICATION
PART2_CLAS_SAMPLE
CLASSIFICATION
PART_CLAS_SAMPLE
CLASSIFICATION
SVMC_SH_CLAS_SAMPLE
CLASSIFICATION
SVMO_SH_CLAS_SAMPLE
```

```
CLASSIFICATION
T_SVM_CLAS_SAMPLE
REGRESSION                SVMR_SH_REGR_SAMPLE
```

The models are created by the following examples:

- PART2_CLAS_SAMPLE by `oml4sql-partitioned-models-svm.sql`
- PART_CLAS_SAMPLE by `oml4sql-partitioned-models-svm.sql`
- SVMC_SH_CLAS_SAMPLE by `oml4sql-classification-svm.sql`
- SVMO_SH_CLAS_SAMPLE by `oml4sql-anomaly-detection-1class-svm.sql`
- T_SVM_CLAS_SAMPLE by `oml4sql-classification-text-mining-svm.sql`
- SVMR_SH_REGR_SAMPLE by `oml4sql-regression-svm.sql`

Related Topics

- ALL_MINING_MODELS

2.2.2 ALL_MINING_MODEL_ATTRIBUTES

Describes an example of `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query.

The following example describes `ALL_MINING_MODEL_ATTRIBUTES` and shows a sample query. Attributes are the predictors or conditions that are used to create models and score data.

Example 2-2 ALL_MINING_MODEL_ATTRIBUTES

```
describe ALL_MINING_MODEL_ATTRIBUTES
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 ATTRIBUTE_NAME                            NOT NULL VARCHAR2(128)
 ATTRIBUTE_TYPE                                     VARCHAR2(11)
 DATA_TYPE                                          VARCHAR2(106)
 DATA_LENGTH                                        NUMBER
 DATA_PRECISION                                     NUMBER
 DATA_SCALE                                         NUMBER
 USAGE_TYPE                                         VARCHAR2(8)
 TARGET                                             VARCHAR2(3)
 ATTRIBUTE_SPEC                                     VARCHAR2(4000)
```

The following query returns the attributes of an SVM classification model named T_SVM_CLAS_SAMPLE. The model has both categorical and numerical attributes and includes one attribute that is unstructured text. The model is created by the `oml4sql-classification-text-mining-svm.sql` example

```
SELECT attribute_name, attribute_type, target
    FROM all_mining_model_attributes
    WHERE model_name = 'T_SVM_CLAS_SAMPLE'
```

```
    ORDER BY attribute_name;
```

The output is as follows:

```
ATTRIBUTE_NAME            ATTRIBUTE_TYPE
TAR
------------------------- --------------------
---
AFFINITY_CARD             CATEGORICAL
YES
AGE                       NUMERICAL
NO
BOOKKEEPING_APPLICATION   NUMERICAL
NO
BULK_PACK_DISKETTES       NUMERICAL
NO
COMMENTS                  TEXT
NO
COUNTRY_NAME              CATEGORICAL
NO
CUST_GENDER               CATEGORICAL
NO
CUST_INCOME_LEVEL         CATEGORICAL
NO
CUST_MARITAL_STATUS       CATEGORICAL
NO
EDUCATION                 CATEGORICAL
NO
FLAT_PANEL_MONITOR        NUMERICAL
NO
HOME_THEATER_PACKAGE      NUMERICAL
NO
HOUSEHOLD_SIZE            CATEGORICAL
NO
OCCUPATION                CATEGORICAL
NO
OS_DOC_SET_KANJI          NUMERICAL
NO
PRINTER_SUPPLIES          NUMERICAL
NO
YRS_RESIDENCE             NUMERICAL
NO
Y_BOX_GAMES               NUMERICAL           NO
```

Related Topics

- ALL_MINING_MODEL_ATTRIBUTES

2.2.3 ALL_MINING_MODEL_PARTITIONS

Describes an example of `ALL_MINING_MODEL_PARTITIONS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_PARTITIONS` and shows a sample query.

Example 2-3 ALL_MINING_MODEL_PARTITIONS

```
describe ALL_MINING_MODEL_PARTITIONS
```

The output is as follows:

```
Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 PARTITION_NAME                                     VARCHAR2(128)
 POSITION                                           NUMBER
 COLUMN_NAME                               NOT NULL VARCHAR2(128)
 COLUMN_VALUE                                       VARCHAR2(4000)
```

The following query returns the partition names and partition key values for two partitioned models. Model PART2_CLAS_SAMPLE has a two column partition key with system-generated partition names. The models are created by the `oml4sql-partitioned-models-svm.sql` example.

```
SELECT model_name, partition_name, position, column_name, column_value
    FROM all_mining_model_partitions
    ORDER BY model_name, partition_name, position;
```

The output is as follows:

```
MODEL_NAME           PARTITION_ POSITION COLUMN_NAME
COLUMN_VALUE
-------------------- ---------- -------- --------------------
--------------
PART2_CLAS_SAMPLE    DM$$_P0           1 CUST_GENDER
F
PART2_CLAS_SAMPLE    DM$$_P0           2 CUST_INCOME_LEVEL
HIGH
PART2_CLAS_SAMPLE    DM$$_P1           1 CUST_GENDER
F
PART2_CLAS_SAMPLE    DM$$_P1           2 CUST_INCOME_LEVEL
LOW
PART2_CLAS_SAMPLE    DM$$_P2           1 CUST_GENDER
F
PART2_CLAS_SAMPLE    DM$$_P2           2 CUST_INCOME_LEVEL
MEDIUM
PART2_CLAS_SAMPLE    DM$$_P3           1 CUST_GENDER
M
PART2_CLAS_SAMPLE    DM$$_P3           2 CUST_INCOME_LEVEL
HIGH
PART2_CLAS_SAMPLE    DM$$_P4           1 CUST_GENDER
M
PART2_CLAS_SAMPLE    DM$$_P4           2 CUST_INCOME_LEVEL
LOW
```

```
PART2_CLAS_SAMPLE    DM$$_P5           1 CUST_GENDER
M
PART2_CLAS_SAMPLE    DM$$_P5           2 CUST_INCOME_LEVEL
MEDIUM
PART_CLAS_SAMPLE     F                 1 CUST_GENDER
F
PART_CLAS_SAMPLE     M                 1 CUST_GENDER
M
PART_CLAS_SAMPLE     U                 1 CUST_GENDER          U
```

Related Topics

- ALL_MINING_MODEL_PARTITIONS

2.2.4 ALL_MINING_MODEL_SETTINGS

Describes an example of `ALL_MINING_MODEL_SETTINGS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_SETTINGS` and shows a sample query. Settings influence model behavior. Settings may be specific to an algorithm or to a machine learning technique, or they may be general.

Example 2-4 ALL_MINING_MODEL_SETTINGS

```
describe ALL_MINING_MODEL_SETTINGS
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 SETTING_NAME                              NOT NULL VARCHAR2(30)
 SETTING_VALUE                                      VARCHAR2(4000)
 SETTING_TYPE                                       VARCHAR2(7)
```

The following query returns the settings for a model named SVD_SH_SAMPLE. The model uses the Singular Value Decomposition algorithm for feature extraction. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

```
SELECT setting_name, setting_value, setting_type
    FROM all_mining_model_settings
    WHERE model_name = 'SVD_SH_SAMPLE'
    ORDER BY setting_name;
```

The output is as follows:

```
SETTING_NAME                   SETTING_VALUE
SETTING
------------------------------ ------------------------------
-------
ALGO_NAME                      ALGO_SINGULAR_VALUE_DECOMP
INPUT
```

```
ODMS_DETAILS                    ODMS_ENABLE                    DEFAULT
ODMS_MISSING_VALUE_TREATMENT    ODMS_MISSING_VALUE_AUTO
DEFAULT
ODMS_SAMPLING                   ODMS_SAMPLING_DISABLE
DEFAULT
PREP_AUTO                       OFF
INPUT
SVDS_SCORING_MODE               SVDS_SCORING_SVD
DEFAULT
SVDS_U_MATRIX_OUTPUT            SVDS_U_MATRIX_ENABLE           INPUT
```

Related Topics

- ALL_MINING_MODEL_SETTINGS

2.2.5 ALL_MINING_MODEL_VIEWS

Describes an example of `ALL_MINING_MODEL_VIEWS` and shows a sample query.

The following example describes `ALL_MINING_MODEL_VIEWS` and shows a sample query. Model views provide details on the models.

Example 2-5 ALL_MINING_MODEL_VIEWS

```
describe ALL_MINING_MODEL_VIEWS
```

The output is as follows:

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 VIEW_NAME                                 NOT NULL VARCHAR2(128)
 VIEW_TYPE                                          VARCHAR2(128)
```

The following query returns the model views for the SVD_SH_SAMPLE model. The model uses the Singular Value Decomposition algorithm for feature extraction. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

```
SELECT view_name, view_type
    FROM all_mining_model_views
    WHERE model_name = 'SVD_SH_SAMPLE'
    ORDER BY view_name;
```

The output is as follows:

```
VIEW_NAME
VIEW_TYPE
-------------------------
--------------------------------------------------
```

```
DM$VESVD_SH_SAMPLE        Singular Value Decomposition S
Matrix
DM$VGSVD_SH_SAMPLE        Global Name-Value
Pairs
DM$VNSVD_SH_SAMPLE        Normalization and Missing Value
Handling
DM$VSSVD_SH_SAMPLE        Computed
Settings
DM$VUSVD_SH_SAMPLE        Singular Value Decomposition U
Matrix
DM$VVSVD_SH_SAMPLE        Singular Value Decomposition V
Matrix
DM$VWSVD_SH_SAMPLE        Model Build Alerts
```

Related Topics

- ALL_MINING_MODEL_VIEWS

2.2.6 ALL_MINING_MODEL_XFORMS

Describes an example of `ALL_MINING_MODEL_XFORMS` and provides a sample query.

The following example describes `ALL_MINING_MODEL_XFORMS` and provides a sample query.

Example 2-6 ALL_MINING_MODEL_XFORMS

```
describe ALL_MINING_MODEL_XFORMS
```

```
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(128)
 MODEL_NAME                                NOT NULL VARCHAR2(128)
 ATTRIBUTE_NAME                                     VARCHAR2(128)
 ATTRIBUTE_SUBNAME                                  VARCHAR2(4000)
 ATTRIBUTE_SPEC                                     VARCHAR2(4000)
 EXPRESSION                                         CLOB
 REVERSE                                            VARCHAR2(3)
```

The following query returns the embedded transformations for a model `PART2_CLAS_SAMPLE`. The model is created by the `oml4sql-partitioned-models-svm.sql` example.

```
SELECT attribute_name, expression
    FROM all_mining_model_xforms
    WHERE model_name = 'PART2_CLAS_SAMPLE'
    ORDER BY attribute_name;
```

The output is as follows:

```
ATTRIBUTE_NAME

------------------------
```

```
EXPRESSION

--------------------------------------------------------------------------------
--
CUST_INCOME_LEVEL

CASE CUST_INCOME_LEVEL WHEN 'A: Below 30,000' THEN
'LOW'
    WHEN 'L: 300,000 and above' THEN
'HIGH'
    ELSE 'MEDIUM' END
```

Related Topics

- ALL_MINING_MODEL_XFORMS

2.3 Oracle Machine Learning Modeling, Transformations, and Convenience Functions

You can access PL/SQL interface to perform data modeling, transformations, and predictive analytics.

The following table displays the PL/SQL packages for Oracle Machine Learning. In Oracle AI Database releases prior to Release 21c, Oracle Machine Learning was named Oracle Data Mining.

Table 2-2 Oracle Machine Learning PL/SQL Packages

Package Name	Description
DBMS_DATA_MINING	Routines for creating and managing machine learning models
DBMS_DATA_MINING_TRANSFORM	Routines for transforming the data for machine learning
DBMS_PREDICTIVE_ANALYTICS	Routines that perform predictive analytics

- DBMS_DATA_MINING
  The DBMS_DATA_MINING package contains routines for creating machine learning models, for performing operations on the models, and for querying them.
- DBMS_DATA_MINING_TRANSFORM
  The DBMS_DATA_MINING_TRANSFORM package contains routines that perform data transformations such as binning, normalization, and outlier treatment.
- DBMS_PREDICTIVE_ANALYTICS
  The DBMS_PREDICTIVE_ANALYTICS package contains routines that perform an automated form of machine learning known as predictive analytics. With predictive analytics, you do not need to be aware of model building or scoring. All machine learning activities are handled internally by the procedure.

Related Topics

- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_PREDICTIVE_ANALYTICS

2.3.1 DBMS_DATA_MINING

The DBMS_DATA_MINING package contains routines for creating machine learning models, for performing operations on the models, and for querying them.

The package includes routines for:

- Creating, dropping, and performing other DDL operations on machine learning models
- Obtaining detailed information about model attributes, rules, and other information internal to the model (model details)
- Computing test metrics for classification models
- Specifying costs for classification models
- Exporting and importing models
- Building models using Oracle Machine Learning native algorithms as well as algorithms written in R

Related Topics

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

2.3.2 DBMS_DATA_MINING_TRANSFORM

The DBMS_DATA_MINING_TRANSFORM package contains routines that perform data transformations such as binning, normalization, and outlier treatment.

The package includes routines for:

- Specifying transformations in a format that can be embedded in a machine learning model.
- Specifying transformations as relational views (external to machine learning model objects).
- Specifying distinct properties for columns in the build data. For example, you can specify that the column must be interpreted as unstructured text, or that the column must be excluded from Automatic Data Preparation.
- Transformation Methods in DBMS_DATA_MINING_TRANSFORM
  Summarizes the methods for transforming data in DBMS_DATA_MINING_TRANSFORM package.

Related Topics

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

2.3.2.1 Transformation Methods in DBMS_DATA_MINING_TRANSFORM

Summarizes the methods for transforming data in DBMS_DATA_MINING_TRANSFORM package.

Table 2-3 DBMS_DATA_MINING_TRANSFORM Transformation Methods

Transformation Method	Description
XFORM interface	CREATE, INSERT, and XFORM routines specify transformations in external views
STACK interface	CREATE, INSERT, and XFORM routines specify transformations for embedding in a model

Table 2-3 (Cont.) DBMS_DATA_MINING_TRANSFORM Transformation Methods

Transformation Method	Description
SET_TRANSFORM	Specifies transformations for embedding in a model

The statements in the following example create a Support Vector Machine (SVM) classification model called T_SVM_Clas_sample with an embedded transformation that causes the comments attribute to be treated as unstructured text data. The T_SVM_CLAS_SAMPLE model is created by `oml4sql-classification-text-mining-svm.sql` example.

Example 2-7 Sample Embedded Transformation

```
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(
    xformlist, 'comments', null, 'comments', null, 'TEXT');
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'T_SVM_Clas_sample',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_build_text',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 't_svmc_sample_settings',
    xform_list => xformlist);
END;
/
```

2.3.3 DBMS_PREDICTIVE_ANALYTICS

The DBMS_PREDICTIVE_ANALYTICS package contains routines that perform an automated form of machine learning known as predictive analytics. With predictive analytics, you do not need to be aware of model building or scoring. All machine learning activities are handled internally by the procedure.

The DBMS_PREDICTIVE_ANALYTICS package includes these routines:

- **EXPLAIN** ranks attributes in order of influence in explaining a target column.
- **PREDICT** predicts the value of a target column based on values in the input data.
- **PROFILE** generates rules that describe the cases from the input data.

The EXPLAIN statement in the following example lists attributes in the view mining_data_build_v in order of their importance in predicting affinity_card.

Example 2-8 Sample EXPLAIN Statement

```
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
        data_table_name      => 'mining_data_build_v',
        explain_column_name  => 'affinity_card',
        result_table_name    => 'explain_results');
END;
/
```

Related Topics

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

2.4 Oracle Machine Learning for SQL Scoring Functions

Use OML4SQL functions score data. Functions can apply a machine learning model schema object to data or dynamically mine it with an analytic clause. SQL functions exist for all OML4SQL scoring algorithms.

All OML4SQL functions, as listed in the following table can operate on an R machine learning model with the corresponding OML4SQL function. However, the functions are not limited to the ones listed here.

Table 2-4 OML4SQL Functions

Function	Description
`CLUSTER_ID`	Returns the ID of the predicted cluster
`CLUSTER_DETAILS`	Returns detailed information about the predicted cluster
`CLUSTER_DISTANCE`	Returns the distance from the centroid of the predicted cluster
`CLUSTER_PROBABILITY`	Returns the probability of a case belonging to a given cluster
`CLUSTER_SET`	Returns a list of all possible clusters to which a given case belongs along with the associated probability of inclusion
`FEATURE_COMPARE`	Compares two similar and dissimilar set of texts from two different documents or keyword phrases or a combination of both
`FEATURE_ID`	Returns the ID of the feature with the highest coefficient value
`FEATURE_DETAILS`	Returns detailed information about the predicted feature
`FEATURE_SET`	Returns a list of objects containing all possible features along with the associated coefficients
`FEATURE_VALUE`	Returns the value of the predicted feature
`ORA_DM_PARTITION_NAME`	Returns the partition names for a partitioned model
`PREDICTION`	Returns the best prediction for the target
`PREDICTION_BOUNDS`	(GLM only) Returns the upper and lower bounds of the interval wherein the predicted values (linear regression) or probabilities (logistic regression) lie.
`PREDICTION_COST`	Returns a measure of the cost of incorrect predictions
`PREDICTION_DETAILS`	Returns detailed information about the prediction

Table 2-4 (Cont.) OML4SQL Functions

Function	Description
PREDICTION_PROBABILITY	Returns the probability of the prediction
PREDICTION_SET	Returns the results of a classification model, including the predictions and associated probabilities for each case
VECTOR_EMBEDDING	Generates a single vector embedding for different data types

The following example shows a query that returns the results of the CLUSTER_ID function. The query applies the model em_sh_clus_sample, which finds groups of customers that share certain characteristics. The query returns the identifiers of the clusters and the number of customers in each cluster. The em_sh_clus_sample model is created by the oml4sql-clustering-expectation-maximization.sql example.

Example 2-9 CLUSTER_ID Function

```
-- -List the clusters into which the customers in this
-- -data set have been grouped.
--
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
  FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;


-- List the clusters into which the customers in this
-- data set have been grouped.
--
SELECT CLUSTER_ID(em_sh_clus_sample USING *) AS clus, COUNT(*) AS cnt
FROM mining_data_apply_v
GROUP BY CLUSTER_ID(em_sh_clus_sample USING *)
ORDER BY cnt DESC;
```

The output is as follows:

```
      CLUS        CNT
---------- ----------
         9        311
         3        294
         7        215
        12        201
        17        123
        16        114
        14         86
        19         64
        15         56
        18         36
```

2.5 Oracle Machine Learning for SQL Statistical Functions

Various SQL statistical functions are available in Oracle AI Database to explore and analyze data.

A variety of scalable statistical functions are accessible through SQL in Oracle AI Database 26ai. These statistical functions are implemented as SQL functions. The SQL statistical functions can be used to compute standard univariate statistics such as `MEAN`, `MAX`, `MIN`, `MEDIAN`, `MODE`, and standard deviation on the data. Users can also perform various other statistical functions such as t-test, f-test, aggregate functions, analytic functions, or ANOVA. The functions listed in the following table are available from SQL.

Table 2-5 SQL Statistical Functions Supported by OML4SQL

Function	Description
`APPROX_COUNT`	Returns approximate count of an expression
`APPROX_SUM`	Returns approximate sum of an expression
`APPROX_RANK`	Returns approximate value in a group of values
`CORR`	Retuns the coefficient of correlation of a set of number pairs
`CORR_S`	Calculates the Spearman's rho correlation coefficient
`CORR_K`	Calculates the Kendall's tau-b correlation coefficient
`COVAR_POP`	Returns the population covariance of a set of number pairs
`COVAR_SAMP`	Returns the sample covariance of a set of number pairs.
`LAG`	`LAG` is an analytic function. It provides access to more than one row of a table at the same time without a self join.
`LEAD`	`LEAD` is an analytic function. It provides access to more than one row of a table at the same time without a self join.
`STATS_BINOMIAL_TEST`	`STATS_BINOMIAL_TEST` is an exact probability test used for dichotomous variables, where only two possible values exist.
`STATS_CROSSTAB`	`STATS_CROSSTAB` is a method used to analyze two nominal variables.
`STATS_F_TEST`	`STATS_F_TEST` tests whether two variances are significantly different.
`STATS_KS_TEST`	`STATS_KS_TEST` is a Kolmogorov-Smirnov function that compares two samples to test whether they are from the same population or from populations that have the same distribution.
`STATS_MODE`	Takes as its argument a set of values and returns the value that occurs with the greatest frequency
`STATS_MW_TEST`	A Mann Whitney test compares two independent samples to test the null hypothesis that two populations have the same distribution function against the alternative hypothesis that the two distribution functions are different.

Table 2-5 (Cont.) SQL Statistical Functions Supported by OML4SQL

Function	Description
STATS_ONE_WAY_ANOVA	Tests differences in means (for groups or variables) for statistical significance by comparing two different estimates of variance
STATS_T_TEST_*	The t-test measures the significance of a difference of means
STATS_T_TEST_ONE	A one-sample t-test
STATS_T_TEST_PAIRED	A two-sample, paired t-test (also known as a crossed t-test)
STATS_T_TEST_INDEP and STATS_T_TEST_INDEPU	A t-test of two independent groups with the same variance (pooled variances) A t-test of two independent groups with unequal variance (unpooled variances)
STDDEV	returns the sample standard deviation of a set of numbers
STDDEV_POP	Computes the population standard deviation and returns the square root of the population variance
STDDEV_SAMP	Computes the cumulative sample standard deviation and returns the square root of the sample variance
SUM	Returns the sum of values

DBMS_STAT_FUNCS PL/SQL package is also available for users.

3
Prepare the Data

Learn how to access and treat the data that can be used to build a model.

- Data Requirements
  Understand how data is stored and viewed for Oracle Machine Learning.
- About Attributes
  Attributes are the items of data that are used in machine learning. Attributes are also referred as variables, fields, or predictors.
- Use Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- Use Market Basket Data
  Understand the use of association and Apriori for market basket analysis.
- Use Retail Data for Analysis
  Retail analysis often makes use of association rules and association models.
- Handle Missing Values
  Understand sparse data and missing values.
- About Transformations
  Understand how you can transform data by using Automatic Data Preparation (ADP) and embedded data transformation.
- Prepare the Case Table
  The first step in preparing data for machine learning is the creation of a case table.

3.1 Data Requirements

Understand how data is stored and viewed for Oracle Machine Learning.

Machine learning activities require data that is defined within a single table or view. The information for each record must be stored in a separate row. The data records are commonly called **cases**. Each case can optionally be identified by a unique **case ID**. The table or view itself can be referred to as a **case table**.

The `CUSTOMERS` table in the `SH` schema is an example of a table that could be used for machine learning. All the information for each customer is contained in a single row. The case ID is the `CUST_ID` column. The rows listed in the following example are selected from `SH.CUSTOMERS`.

> **Note**
>
> Oracle Machine Learning requires single-record case data for all types of models except association models, which can be built on native transactional data.

Example 3-1 Sample Case Table

```
select cust_id, cust_gender, cust_year_of_birth,
          cust_main_phone_number from sh.customers where cust_id < 11;
```

The output is as follows:

```
CUST_ID CUST_GENDER CUST_YEAR_OF_BIRTH CUST_MAIN_PHONE_NUMBER
------- ----------- ---- ------------- ------------------------
1        M               1946          127-379-8954
2        F               1957          680-327-1419
3        M               1939          115-509-3391
4        M               1934          577-104-2792
5        M               1969          563-667-7731
6        F               1925          682-732-7260
7        F               1986          648-272-6181
8        F               1964          234-693-8728
9        F               1936          697-702-2618
10       F               1947          601-207-4099
```

- Column Data Types
  Understand the different types of column data in a case table.
- Vector Data Type
  You can provide `VECTOR` data as input to Oracle Machine Learning in-database algorithms to complement other structured data or be used alone. The vector data type is supported for clustering, classification, anomaly detection, and feature extraction.
- Data Sets for Classification and Regression
  Understand how data sets are used for training and testing the model.
- Scoring Requirements
  Learn how scoring is done in Oracle Machine Learning for SQL.

Related Topics

- Use Market Basket Data
  Understand the use of association and Apriori for market basket analysis.

3.1.1 Column Data Types

Understand the different types of column data in a case table.

The columns of the case table hold the attributes that describe each case. In Example 3-1, the attributes are: `CUST_GENDER`, `CUST_YEAR_OF_BIRTH`, and `CUST_MAIN_PHONE_NUMBER`. The attributes are the predictors in a supervised model or the descriptors in an unsupervised model. The case ID, `CUST_ID`, can be viewed as a special attribute; it is not a predictor or a descriptor.

OML4SQL supports standard Oracle data types except `DATE`, `TIMESTAMP`, `RAW`, and `LONG`. Oracle Machine Learning supports date type (datetime, date, timestamp) for case_id, `CLOB`/`BLOB`/`FILE` that are interpreted as text columns, and the following collection types as well:

```
DM_NESTED_CATEGORICALS
DM_NESTED_NUMERICALS
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
```

> **Note**
>
> The attributes with the data type `BOOLEAN` are treated as numeric with the following values: `TRUE` means `1`, `FALSE` means `0`, and `NULL` is interpreted as an unknown value. The `CASE_ID_COLUMN_NAME` attribute does not support `BOOLEAN` data type.

Related Topics

- Use Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- About Unstructured Text
  Unstructured text may contain important information that is critical to the success of a business.
- *Oracle Database SQL Language Reference*

3.1.2 Vector Data Type

You can provide `VECTOR` data as input to Oracle Machine Learning in-database algorithms to complement other structured data or be used alone. The vector data type is supported for clustering, classification, anomaly detection, and feature extraction.

While dense vectors with arbitrary precision and dimensions are supported, in a flex vector column, precision may differ and dimensions within a single vector column must remain consistent. Errors are raised for mismatched dimensions.

Partitioned models track vector dimensions alongside partition statistics. Different partitions can have different vector dimensions, however, dimensions must remain consistent within a single partition. Errors are raised for mismatched dimensions within a single partition.

The system supports `FLOAT32`, `FLOAT64`, and `INT8` as datatypes. Vectors with `FLEX` dimension and precision are supported. These features can be used in combination with the other data types supported by OML (numerical, categorical, nested, and text).

Scoring with Vectors

The system treats each vector dimension as an individual predictor and provides model details at the vector component level, labeled as `DM$$VECxxx`, where `xxx` represents the component's position. For example, `DM$$VEC1`. During scoring, the system matches vector dimensions between the model and input data at compile time or runtime, raising errors if mismatches occur. A vector cannot be a target or a `case_id` column, errors are raised if you set vector as a target or `case_id`.

The system does not support:

- analytic scoring with vectors, analytic scoring operators skip the vector inputs without displaying any error.
- sparse vectors, raises an error that the format is not supported if sparse vectors are identified. To learn more about sparse vectors, see Create Tables Using the VECTOR Data Type.
- binary vector precision, raises an error that the format is not supported

OML supports the vector data type for the following algorithms and the scoring operators:

Technique	Algorithms	Scoring Operator
Classification or Regression	SVM, Neural Network, GLM	`PREDICTION`, `PREDICTION_PROBABILITY`, `PREDICTION_SET`, `PREDICTION_BOUNDS`
Anomaly Detection	One-class SVM, Expectation Maximization	`PREDICTION`, `PREDICTION_PROBABILITY`, `PREDICTION_SET`
Clustering	*k*-Means, Expectation Maximization	`CLUSTER_ID`, `CLUSTER_PROBABILITY`, `CLUSTER_SET`, `CLUSTER_DISTANCE`
Feature Extraction	SVD, PCA	`FEATURE_ID`, `FEATURE_VALUE`, `FEATURE_SET`, `VECTOR_EMBEDDING`

See Example: Using Vector Data for Dimensionality Reduction and Clustering for more details.

3.1.3 Data Sets for Classification and Regression

Understand how data sets are used for training and testing the model.

You need two case tables to build and validate classification and regression models. One set of rows is used for training the model, another set of rows is used for testing the model. It is often convenient to derive the build data and test data from the same data set. For example, you could randomly select 60% of the rows for training the model; the remaining 40% could be used for testing the model.

Models that implement other machine learning functions, such as attribute importance, clustering, association, or feature extraction, do not use separate test data.

3.1.4 Scoring Requirements

Learn how scoring is done in Oracle Machine Learning for SQL.

Most machine learning models can be applied to separate data in a process known as **scoring**. Oracle Machine Learning for SQL supports the scoring operation for classification, regression, anomaly detection, clustering, and feature extraction.

The scoring process matches column names in the scoring data with the names of the columns that were used to build the model. The scoring process does not require all the columns to be present in the scoring data. If the data types do not match, OML4SQL attempts to perform type coercion. For example, if a column called `PRODUCT_RATING` is `VARCHAR2` in the training data but `NUMBER` in the scoring data, OML4SQL effectively applies a `TO_CHAR()` function to convert it.

The column in the test or scoring data must undergo the same transformations as the corresponding column in the build data. For example, if the `AGE` column in the build data was transformed from numbers to the values `CHILD`, `ADULT`, and `SENIOR`, then the `AGE` column in the scoring data must undergo the same transformation so that the model can properly evaluate it.

> **Note**
>
> OML4SQL can embed user-specified transformation instructions in the model and reapply them whenever the model is applied. When the transformation instructions are embedded in the model, you do not need to specify them for the test or scoring data sets.
>
> OML4SQL also supports Automatic Data Preparation (ADP). When ADP is enabled, the transformations required by the algorithm are performed automatically and embedded in the model along with any user-specified transformations.

> **See Also**
>
> Automatic Data Preparation and Embed Transformations in a Model for more information on automatic and embedded data transformations

3.2 About Attributes

Attributes are the items of data that are used in machine learning. Attributes are also referred as variables, fields, or predictors.

In predictive models, attributes are the predictors that affect a given outcome. In descriptive models, attributes are the items of information being analyzed for natural groupings or associations. For example, a table of employee data that contains attributes such as job title, date of hire, salary, age, gender, and so on.

- Data Attributes and Model Attributes
  Data attributes are columns in the data set used to build, test, or score a model. **Model attributes** are the data representations used internally by the model.
- Target Attribute
  Understand what a **target** means in machine learning and understand the different target data types.
- Numericals, Categoricals, and Unstructured Text
  Explains numeric, categorical, and unstructured text attributes.
- Model Signature
  Learn about model signature and the data types that are considered in the build data.
- Scoping of Model Attribute Name
  Learn about model attribute name.
- Model Details
  Model details reveal information about model attributes and their treatment by the algorithm. Oracle recommends that users leverage the model detail views for the respective algorithm.

3.2.1 Data Attributes and Model Attributes

Data attributes are columns in the data set used to build, test, or score a model. **Model attributes** are the data representations used internally by the model.

Data attributes and model attributes can be the same. For example, a column called SIZE, with values S, M, and L, are attributes used by an algorithm to build a model. Internally, the model attribute SIZE is most likely be the same as the data attribute from which it was derived.

On the other hand, a nested column SALES_PROD, containing the sales figures for a group of products, does not correspond to a model attribute. The data attribute can be SALES_PROD, but each product with its corresponding sales figure (each row in the nested column) is a model attribute.

Transformations also cause a discrepancy between data attributes and model attributes. For example, a transformation can apply a calculation to two data attributes and store the result in a new attribute. The new attribute is a model attribute that has no corresponding data attribute. Other transformations such as binning, normalization, and outlier treatment, cause the model's representation of an attribute to be different from the data attribute in the case table.

Related Topics

- Use Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to DBMS_DATA_MINING.CREATE_MODEL2 or DBMS_DATA_MINING.CREATE_MODEL.

3.2.2 Target Attribute

Understand what a **target** means in machine learning and understand the different target data types.

The **target** of a supervised model is a special kind of attribute. The target column in the training data contains the historical values used to train the model. The target column in the test data contains the historical values to which the predictions are compared. The act of scoring produces a prediction for the target.

Clustering, feature extraction, association, and anomaly detection models do not use a target.

Nested columns and columns of unstructured data (such as BFILE, CLOB, or BLOB) cannot be used as targets.

Table 3-1 Target Data Types

Machine Learning Function	Target Data Types
Classification	VARCHAR2, CHAR
	NUMBER, FLOAT
	BINARY_DOUBLE, BINARY_FLOAT, ORA_MINING_VARCHAR2_NT
	BOOLEAN
Regression	NUMBER, FLOAT
	BINARY_DOUBLE, BINARY_FLOAT

You can query the *_MINING_MODEL_ATTRIBUTES view to find the target for a given model.

Related Topics

- ALL_MINING_MODEL_ATTRIBUTES
  Describes an example of ALL_MINING_MODEL_ATTRIBUTES and shows a sample query.

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

3.2.3 Numericals, Categoricals, and Unstructured Text

Explains numeric, categorical, and unstructured text attributes.

Model attributes are numerical, categorical, or unstructured (text). Data attributes, which are columns in a case table, have Oracle data types, as described in "Column Data Types".

Numerical attributes can theoretically have an infinite number of values. The values have an implicit order, and the differences between them are also ordered. Oracle Machine Learning for SQL interprets `NUMBER`, `FLOAT`, `BINARY_DOUBLE`, `BINARY_FLOAT`, `BOOLEAN`, `DM_NESTED_NUMERICALS`, `DM_NESTED_BINARY_DOUBLES`, and `DM_NESTED_BINARY_FLOATS` as numerical.

Categorical attributes have values that identify a finite number of discrete categories or classes. There is no implicit order associated with the values. Some categoricals are binary: they have only two possible values, such as yes or no, or male or female. Other categoricals are multi-class: they have more than two values, such as small, medium, and large.

OML4SQL interprets `CHAR` and `VARCHAR2` as categorical by default, however these columns may also be identified as columns of unstructured data (text). OML4SQL interprets columns of `DM_NESTED_CATEGORICALS` as categorical. Columns of `CLOB`, `BLOB`, and `BFILE` always contain unstructured data.

The target of a classification model is categorical. (If the target of a classification model is numeric, it is interpreted as categorical.) The target of a regression model is numerical. The target of an attribute importance model is either categorical or numerical.

Related Topics

- Column Data Types
  Understand the different types of column data in a case table.
- About Unstructured Text
  Unstructured text may contain important information that is critical to the success of a business.

3.2.4 Model Signature

Learn about model signature and the data types that are considered in the build data.

The model signature is the set of data attributes that are used to build a model. Some or all of the attributes in the signature must be present for scoring. The model accounts for any missing columns on a best-effort basis. If columns with the same names but different data types are present, the model attempts to convert the data type. If extra, unused columns are present, they are disregarded.

The model signature does not necessarily include all the columns in the build data. Algorithm-specific criteria can cause the model to ignore certain columns. Other columns can be eliminated by transformations. Only the data attributes actually used to build the model are included in the signature.

The target and case ID columns are not included in the signature.

3.2.5 Scoping of Model Attribute Name

Learn about model attribute name.

The model attribute name consists of two parts: a column name, and a subcolumn name.

```
column_name[.subcolumn_name]
```

The `column_name` component is the name of the data attribute. It is present in all model attribute names. Nested attributes and text attributes also have a `subcolumn_name` component as shown in the following example.

Example 3-2 Model Attributes Derived from a Nested Column

The nested column `SALESPROD` has three rows.

```
SALESPROD(ATTRIBUTE_NAME, VALUE)
--------------------------------
((PROD1, 300),
 (PROD2, 245),
 (PROD3, 679))
```

The name of the data attribute is `SALESPROD`. Its associated model attributes are:

```
SALESPROD.PROD1
SALESPROD.PROD2
SALESPROD.PROD3
```

3.2.6 Model Details

Model details reveal information about model attributes and their treatment by the algorithm. Oracle recommends that users leverage the model detail views for the respective algorithm.

Transformation and reverse transformation expressions are associated with model attributes. Transformations are applied to the data attributes before the algorithmic processing that creates the model. Reverse transformations are applied to the model attributes after the model has been built, so that the model details are expressed in the form of the original data attributes, or as close to it as possible.

Reverse transformations support model transparency. They provide a view of the data that the algorithm is working with internally but in a format that is meaningful to a user.

Deprecated `GET_MODEL_DETAILS`

There is a separate `GET_MODEL_DETAILS` routine for each algorithm. Starting from Oracle Database 12*c* Release 2, the `GET_MODEL_DETAILS` are deprecated. Oracle recommends to use Model Detail Views for the respective algorithms.

Related Topics

- Model Detail Views

3.3 Use Nested Data

A join between the tables for one-to-many relationship is represented through nested columns.

Oracle Machine Learning for SQL requires a case table in single-record case format, with each record in a separate row. What if some or all of your data is in multi-record case format, with each record in several rows? What if you want one attribute to represent a series or collection of values, such as a student's test scores or the products purchased by a customer?

This kind of one-to-many relationship is usually implemented as a join between tables. For example, you can join your customer table to a sales table and thus associate a list of products purchased with each customer.

OML4SQL supports dimensioned data through nested columns. To include dimensioned data in your case table, create a view and cast the joined data to one of the machine learning nested table types. Each row in the nested column consists of an attribute name/value pair. OML4SQL internally processes each nested row as a separate attribute.

> **Note**
>
> O-Cluster is the only algorithm that does not support nested data.

- Nested Object Types
  Nested tables are object data types that can be used in place of other data types.
- Example: Transforming Transactional Data for Machine Learning
  In this example, a comparison is shown for sale of products in four regions with data before transformation and then after transformation.

Related Topics

- Example: Creating a Nested Column for Market Basket Analysis
  The example shows how to define a nested column for market basket analysis.

3.3.1 Nested Object Types

Nested tables are object data types that can be used in place of other data types.

Oracle AI Database supports user-defined data types that make it possible to model real-world entities as objects in the database. **Collection types** are object data types for modeling multi-valued attributes. Nested tables are collection types. Nested tables can be used anywhere that other data types can be used.

OML4SQL supports the following nested object types:

```
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
DM_NESTED_NUMERICALS
DM_NESTED_CATEGORICALS
```

Descriptions of the nested types are provided in this example.

Example 3-3 OML4SQL Nested Data Types

```
describe dm_nested_binary_double
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_DOUBLE

describe dm_nested_binary_doubles
 DM_NESTED_BINARY_DOUBLES TABLE OF SYS.DM_NESTED_BINARY_DOUBLE
 Name                                       Null?    Type
 ----------------------------------------- --------
--------------------------
 ATTRIBUTE_NAME                                      VARCHAR2(4000)
 VALUE                                               BINARY_DOUBLE
```

```
describe dm_nested_binary_float
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_FLOAT

describe dm_nested_binary_floats
 DM_NESTED_BINARY_FLOATS TABLE OF SYS.DM_NESTED_BINARY_FLOAT
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              BINARY_FLOAT

describe dm_nested_numerical
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              NUMBER

describe dm_nested_numericals
 DM_NESTED_NUMERICALS TABLE OF SYS.DM_NESTED_NUMERICAL
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              NUMBER

describe dm_nested_categorical
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              VARCHAR2(4000)

describe dm_nested_categoricals
 DM_NESTED_CATEGORICALS TABLE OF SYS.DM_NESTED_CATEGORICAL
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 ATTRIBUTE_NAME                                     VARCHAR2(4000)
 VALUE                                              VARCHAR2(4000)
```

Related Topics

- *Oracle Database Object-Relational Developer's Guide*

3.3.2 Example: Transforming Transactional Data for Machine Learning

In this example, a comparison is shown for sale of products in four regions with data before transformation and then after transformation.

Example 3-4 shows data from a view of a sales table. It includes sales for three of the many products sold in four regions. This data is not suitable for machine learning at the product level because sales for each case (product), is stored in several rows.

Example 3-5 shows how this data can be transformed for machine learning. The case ID column is `PRODUCT`. `SALES_PER_REGION`, a nested column of type `DM_NESTED_NUMERICALS`, is a data attribute. This table is suitable for machine learning at the product case level, because the information for each case is stored in a single row.

Oracle Machine Learning for SQL treats each nested row as a separate model attribute, as shown in Example 3-6.

> **Note**
>
> The presentation in this example is conceptual only. The data is not actually pivoted before being processed.

Example 3-4 Product Sales per Region in Multi-Record Case Format

```
PRODUCT   REGION         SALES
-------   --------   ----------
Prod1       NE           556432
Prod2       NE           670155
Prod3       NE             3111
.
.
Prod1       NW            90887
Prod2       NW           100999
Prod3       NW           750437
.
.
Prod1       SE            82153
Prod2       SE            57322
Prod3       SE            28938
.
.
Prod1       SW          3297551
Prod2       SW          4972019
Prod3       SW           884923
.
.
```

Example 3-5 Product Sales per Region in Single-Record Case Format

```
PRODUCT      SALES_PER_REGION
          (ATTRIBUTE_NAME, VALUE)
------    -------------------------
```

```
              Prod1      ('NE' ,     556432)
                         ('NW' ,      90887)
                         ('SE' ,      82153)
                         ('SW' ,    3297551)
              Prod2      ('NE' ,     670155)
                         ('NW' ,     100999)
                         ('SE' ,      57322)
                         ('SW' ,    4972019)
              Prod3      ('NE' ,       3111)
                         ('NW' ,     750437)
                         ('SE' ,      28938)
                         ('SW' ,     884923)
              .
              .
```

Example 3-6 Model Attributes Derived From SALES_PER_REGION

```
PRODUCT    SALES_PER_REGION.NE    SALES_PER_REGION.NW    SALES_PER_REGION.SE
SALES_PER_REGION.SW
-------    -------------------    -------------------    -------------------
-------------------
Prod1                  556432                  90887                  82153
3297551
Prod2                  670155                 100999                  57322
4972019
Prod3                    3111                 750437                  28938
884923
.
.
```

3.4 Use Market Basket Data

Understand the use of association and Apriori for market basket analysis.

Market basket data identifies the items sold in a set of baskets or transactions. Oracle Machine Learning for SQL provides the association machine learning function for market basket analysis.

Association models use the Apriori algorithm to generate association rules that describe how items tend to be purchased in groups. For example, an association rule can assert that people who buy peanut butter are 80% likely to also buy jelly.

Market basket data is usually **transactional**. In transactional data, a case is a transaction and the data for a transaction is stored in multiple rows. OML4SQL association models can be built on transactional data or on single-record case data. The `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` settings specify whether the data for association rules is in transactional format.

> **Note**
>
> Association models are the only type of model that can be built on native transactional data. For all other types of models, OML4SQL requires that the data be presented in single-record case format.

The Apriori algorithm assumes that the data is transactional and that it has many missing values. Apriori interprets all missing values as sparse data, and it has its own native mechanisms for handling sparse data.

- Example: Creating a Nested Column for Market Basket Analysis
  The example shows how to define a nested column for market basket analysis.

> **See Also**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information on the `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` settings.

3.4.1 Example: Creating a Nested Column for Market Basket Analysis

The example shows how to define a nested column for market basket analysis.

Association models can be built on native transactional data or on nested data. The following example shows how to define a nested column for market basket analysis.

The following SQL statement transforms this data to a column of type `DM_NESTED_NUMERICALS` in a view called `SALES_TRANS_CUST_NESTED`. This view can be used as a case table for machine learning.

```
CREATE VIEW sales_trans_cust_nested AS
            SELECT trans_id,
                   CAST(COLLECT(DM_NESTED_NUMERICAL(
                   prod_name, 1))
                   AS DM_NESTED_NUMERICALS) custprods
                FROM sales_trans_cust
            GROUP BY trans_id;
```

This query returns two rows from the transformed data.

```
SELECT * FROM sales_trans_cust_nested
              WHERE trans_id < 101000
              AND trans_id > 100997;
```

The output is as follows:

```
TRANS_ID  CUSTPRODS(ATTRIBUTE_NAME, VALUE)
-------  ------------------------------------------------
100998   DM_NESTED_NUMERICALS
          (DM_NESTED_NUMERICAL('O/S Documentation Set - English', 1)
100999   DM_NESTED_NUMERICALS
          (DM_NESTED_NUMERICAL('CD-RW, High Speed Pack of 5', 1),
           DM_NESTED_NUMERICAL('External 8X CD-ROM', 1),
           DM_NESTED_NUMERICAL('SIMM- 16MB PCMCIAII card', 1))
```

Example 3-7 Convert to a Nested Column

The view `SALES_TRANS_CUST` provides a list of transaction IDs to identify each market basket and a list of the products in each basket.

```
describe sales_trans_cust
```

The output is as follows:

```
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 TRANS_ID                                              NOT NULL NUMBER
 PROD_NAME                                             NOT NULL VARCHAR2(50)
 QUANTITY                                                       NUMBER
```

Related Topics

- Handle Missing Values
  Understand sparse data and missing values.

3.5 Use Retail Data for Analysis

Retail analysis often makes use of association rules and association models.

The association rules are enhanced to calculate aggregates along with rules or itemsets.

- Example: Calculating Aggregates
  This example shows how to calculate aggregates using the customer grocery purchase and profit data.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

3.5.1 Example: Calculating Aggregates

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

Calculating Aggregates for Grocery Store Data

Assume a grocery store has the following data:

Table 3-2 Grocery Store Data

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

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

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

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

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

Related Topics

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

3.6 Handle Missing Values

Understand sparse data and missing values.

Oracle Machine Learning for SQL distinguishes between **sparse data** and data that contains **random missing values**. The latter means that some attribute values are unknown. Sparse data, on the other hand, contains values that are assumed to be known, although they are not represented in the data.

A typical example of sparse data is market basket data. Out of hundreds or thousands of available items, only a few are present in an individual case (the basket or transaction). All the item values are known, but they are not all included in the basket. Present values have a quantity, while the items that are not represented are sparse (with a known quantity of zero).

OML4SQL interprets missing data as follows:

- Missing at random: Missing values in columns with a simple data type (not nested) are assumed to be missing at random.
- Sparse: Missing values in nested columns indicate sparsity.
- Missing Values or Sparse Data?
  Some real life examples are described to interpret missing values and sparse data.
- Missing Value Treatment in Oracle Machine Learning for SQL
  Summarizes the treatment of missing values in OML4SQL.
- Changing the Missing Value Treatment
  Transform the missing data as sparse or missing at random.

3.6.1 Missing Values or Sparse Data?

Some real life examples are described to interpret missing values and sparse data.

The examples illustrate how Oracle Machine Learning for SQL identifies data as either sparse or missing at random.

- Sparsity in a Sales Table
  Understand how Oracle Machine Learning for SQL interprets missing data in nested column.
- Missing Values in a Table of Customer Data
  When the data is not available for some attributes, those missing values are considered to be missing at random.

3.6.1.1 Sparsity in a Sales Table

Understand how Oracle Machine Learning for SQL interprets missing data in nested column.

A sales table contains point-of-sale data for a group of products that are sold in several stores to different customers over a period of time. A particular customer buys only a few of the products. The products that the customer does not buy do not appear as rows in the sales table.

If you were to figure out the amount of money a customer has spent for each product, the unpurchased products have an inferred amount of zero. The value is not random or unknown; it is zero, even though no row appears in the table.

Note that the sales data is dimensioned (by product, stores, customers, and time) and are often represented as nested data for machine learning.

Since missing values in a nested column always indicate sparsity, you must ensure that this interpretation is appropriate for the data that you want to mine. For example, when trying to mine a multi-record case data set containing movie ratings from users of a large movie database, the missing ratings are unknown (missing at random), but Oracle Machine Learning for SQL treats the data as sparse and infer a rating of zero for the missing value.

3.6.1.2 Missing Values in a Table of Customer Data

When the data is not available for some attributes, those missing values are considered to be missing at random.

A table of customer data contains demographic data about customers. The case ID column is the customer ID. The attributes are age, education, profession, gender, house-hold size, and so on. Not all the data is available for each customer. Any missing values are considered to be missing at random. For example, if the age of customer 1 and the profession of customer 2 are not present in the data, that information is unknown. It does not indicate sparsity.

Note that the customer data is not dimensioned. There is a one-to-one mapping between the case and each of its attributes. None of the attributes are nested.

3.6.2 Missing Value Treatment in Oracle Machine Learning for SQL

Summarizes the treatment of missing values in OML4SQL.

Missing value treatment depends on the algorithm and on the nature of the data (categorical or numerical, sparse or missing at random). Missing value treatment is summarized in the following table.

> **Note**
>
> OML4SQL performs the same missing value treatment whether or not you are using Automatic Data Preparation (ADP).

Table 3-3 Missing Value Treatment by Algorithm

Missing Data	EM, GLM, NMF, k-Means, SVD, SVM	DT, MDL, NB, OC	Apriori
NUMERICAL missing at random	The algorithm replaces missing numerical values with the mean. For Expectation Maximization (EM), the replacement only occurs in columns that are modeled with Gaussian distributions.	The algorithm handles missing values naturally as missing at random.	The algorithm interprets all missing data as sparse.
CATEGORICAL missing at random	Generalized Linear Model (GLM), Non-Negative Matrix Factorization (NMF), *k*-Means, and Support Vector Machine (SVM) replaces missing categorical values with the mode. Singular Value Decomposition (SVD) does not support categorical data. EM does not replace missing categorical values. EM treats NULLs as a distinct value with its own frequency count.	The algorithm handles missing values naturally as missing random.	The algorithm interprets all missing data as sparse.
NUMERICAL sparse	The algorithm replaces sparse numerical data with zeros.	O-Cluster does not support nested data and therefore does not support sparse data. Decision Tree (DT), Minimum Description Length (MDL), and Naive Bayes (NB) replace sparse numerical data with zeros.	The algorithm handles sparse data.
CATEGORICAL sparse	All algorithms except SVD replace sparse categorical data with zero vectors. SVD does not support categorical data.	O-Cluster does not support nested data and therefore does not support sparse data. DT, MDL, and NB replace sparse categorical data with the special value `DM$SPARSE`.	The algorithm handles sparse data.

3.6.3 Changing the Missing Value Treatment

Transform the missing data as sparse or missing at random.

If you want Oracle Machine Learning for SQL to treat missing data as sparse instead of missing at random or missing at random instead of sparse, transform it before building the model.

If you want missing values to be treated as sparse, but OML4SQL interprets them as missing at random, you can use a SQL function like `NVL` to replace the nulls with a value such as "NA". OML4SQL does not perform missing value treatment when there is a specified value.

If you want missing nested attributes to be treated as missing at random, you can transform the nested rows into physical attributes in separate columns — as long as the case table stays

within the column limitation imposed by the Database. Fill in all of the possible attribute names, and specify them as null. Alternatively, insert rows in the nested column for all the items that are not present and assign a value such as the mean or mode to each one.

Related Topics

- *Oracle Database SQL Language Reference*

3.7 About Transformations

Understand how you can transform data by using Automatic Data Preparation (ADP) and embedded data transformation.

A transformation is a SQL expression that modifies the data in one or more columns. Data must typically undergo certain transformations before it can be used to build a model. Many Oracle Machine Learning algorithms have specific transformation requirements. Before data can be scored, it must be transformed in the same way that the training data was transformed.

Oracle Machine Learning for SQL supports ADP, which automatically implements the transformations required by the algorithm. The transformations are embedded in the model and automatically run whenever the model is applied.

If additional transformations are required, you can specify them as SQL expressions and supply them as input when you create the model. These transformations are embedded in the model as they are with ADP.

With automatic and embedded data transformation, most of the work of data preparation is handled for you. You can create a model and score multiple data sets in a few steps:

1. Identify the columns to include in the case table.
2. Create nested columns if you want to include transactional data.
3. Write SQL expressions for any transformations not handled by ADP.
4. Create the model, supplying the SQL expressions (if specified) and identifying any columns that contain text data.
5. Ensure that some or all of the columns in the scoring data have the same name and type as the columns used to train the model.

Related Topics

- Scoring Requirements
  Learn how scoring is done in Oracle Machine Learning for SQL.

> **See Also**
>
> OML provides algorithm-specific automatic data preparation and other model building-related features

3.8 Prepare the Case Table

The first step in preparing data for machine learning is the creation of a case table.

If all the data resides in a single table and all the information for each case (record) is included in a single row (single-record case), this process is already taken care of. If the data resides in

several tables, creating the data source involves the creation of a view. For the sake of simplicity, the term "case table" is used here to refer to either a table or a view.

- Convert Column Data Types
  In OML, string columns are treated as categorical, number columns as numerical, and `BOOLEAN` columns are treated as numerical. If you have a numeric column that you want to be treated as a categorical, you must convert it to a string. For example, the day number of the week.
- Extract Datetime Column Values
  You can extract values from a datatime or interval value using the `EXTRACT` function.
- Text Transformation
  Learn text processing using Oracle Machine Learning for SQL.
- About Business and Domain-Sensitive Transformations
  Understand why you need to transform data according to business problems.
- Create Nested Columns
  In transactional data, the information for each case is contained in multiple rows. When the data source includes transactional data (multi-record case), the transactions must be aggregated to the case level in nested columns.

3.8.1 Convert Column Data Types

In OML, string columns are treated as categorical, number columns as numerical, and `BOOLEAN` columns are treated as numerical. If you have a numeric column that you want to be treated as a categorical, you must convert it to a string. For example, the day number of the week.

For example, zip codes identify different postal zones; they do not imply order. If the zip codes are stored in a numeric column, they are interpreted as a numeric attribute. You must convert the data type so that the column data can be used as a categorical attribute by the model. You can do this using the `TO_CHAR` function to convert the digits 1-9 and the `LPAD` function to retain the leading 0, if there is one.

```
LPAD(TO_CHAR(ZIPCODE),5,'0')
```

The attributes with the data type `BOOLEAN` are treated as numeric with the following values: `TRUE` means `1`, `FALSE` means `0`, and `NULL` is interpreted as an unknown value. The `CASE_ID_COLUMN_NAME` attribute does not support `BOOLEAN` data type.

3.8.2 Extract Datetime Column Values

You can extract values from a datatime or interval value using the `EXTRACT` function.

The `EXTRACT` function extracts and returns the value of a specified datetime field from a datetime or interval value expression. The values that can be extracted are `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `TIMEZONE_HOUR`, `TIMEZONE_MINUTE`, `TIMEZONE_REGION`, and `TIMEZONE_ABBR`.

```
sales_tssales_tsCUST_IDTIME_STAMP

select cust_id, time_stamp,
    extract(year from time_stamp) year,
    extract(month from time_stamp) month,
    extract(day from time_stamp) day_of_month,
    to_char(time_stamp,'ww') week_of_year,
```

```
    to_char(time_stamp,'D') day_of_week,
    extract(hour from time_stamp) hour,
    extract(minute from time_stamp) minute,
    extract(second from time_stamp) second
from sales_ts
```

3.8.3 Text Transformation

Learn text processing using Oracle Machine Learning for SQL.

You can use OML4SQL to process text. Columns of text in the case table can be processed once they have undergone the proper transformation.

The text column must be in a table, not a view. The transformation process uses several features of Oracle Text; it treats the text in each row of the table as a separate document. Each document is transformed to a set of text tokens known as **terms**, which have a numeric value and a text label. The text column is transformed to a nested column of `DM_NESTED_NUMERICALS`.

3.8.4 About Business and Domain-Sensitive Transformations

Understand why you need to transform data according to business problems.

Some transformations are dictated by the definition of the business problem. For example, you want to build a model to predict high-revenue customers. Since your revenue data for current customers is in dollars you need to define what "high-revenue" means. Using some formula that you have developed from past experience, you can recode the revenue attribute into ranges Low, Medium, and High before building the model.

Another common business transformation is the conversion of date information into elapsed time. For example, date of birth can be converted to age.

Domain knowledge can be very important in deciding how to prepare the data. For example, some algorithms produce unreliable results if the data contains values that fall far outside of the normal range. In some cases, these values represent errors or unusualities. In others, they provide meaningful information.

Related Topics

- Outlier Treatment
  Understand what you must do to treat outliers.

3.8.5 Create Nested Columns

In transactional data, the information for each case is contained in multiple rows. When the data source includes transactional data (multi-record case), the transactions must be aggregated to the case level in nested columns.

An example is sales data in a star schema when machine learning at the product level. Sales is stored in many rows for a single product (the case) because the product is sold in many stores to many customers over a period of time.

See Also

Using Nested Data for information about converting transactional data to nested columns

4

Create a Model

Explains how to create Oracle Machine Learning for SQL models and to query model details.

- Before Creating a Model
  Explains the preparation steps before creating a model.
- Choose the Machine Learning Technique
  Describes providing an Oracle Machine Learning for SQL machine learning function for the `CREATE_MODEL` and `CREATE_MODEL2` procedure.
- Choose the Algorithm
  Learn about providing the algorithm settings for a model.
- Automatic Data Preparation
  Most algorithms require some form of data transformation. During the model build process, Oracle Machine Learning for SQL can automatically perform the transformations required by the algorithm.
- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.
- The CREATE_MODEL2 Procedure
  The `CREATE_MODEL2` procedure of the `DBMS_DATA_MINING` package is a procedure for defining model settings to build a model.
- The CREATE_MODEL Procedure
  The `CREATE_MODEL` procedure of the `DBMS_DATA_MINING` package uses the specified data to create a machine learning model with the specified name and machine learning function.
- Specify Model Settings
  You can configure your model by specifying model settings.
- Model Detail Views

4.1 Before Creating a Model

Explains the preparation steps before creating a model.

Models are database schema objects that perform machine learning. The `DBMS_DATA_MINING` PL/SQL package is the API for creating, configuring, evaluating, and querying machine learning models (model details).

Before you create a model, you must decide what you want the model to do. You must identify the training data and determine if transformations are required. You can specify model settings to influence the behavior of the model behavior. The preparation steps are summarized in the following table.

Table 4-1 Preparation for Creating an Oracle Machine Learning for SQL Model

Preparation Step	Description
Choose the machine learning function	See Choose the Machine Learning Technique
Choose the algorithm	See Choose the Algorithm
Identify the build (training) data	See Prepare the Data
For classification and regression models, identify the test data	See Data Sets for Classification and Regression
Determine your data transformation strategy and create and populate a settings tables (if needed)	See Specify Model Settings

Related Topics

- About Oracle Machine Learning Models
  Machine learning models are database schema objects that perform machine learning techniques.
- DBMS_DATA_MINING
  The `DBMS_DATA_MINING` package contains routines for creating machine learning models, for performing operations on the models, and for querying them.

4.2 Choose the Machine Learning Technique

Describes providing an Oracle Machine Learning for SQL machine learning function for the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

An OML4SQL machine learning technique specifies a class of problems that can be modeled and solved. You specify a machine learning with the `mining_function` argument of the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

OML4SQL machine learning functions implement either **supervised** or **unsupervised** learning. Supervised learning uses a set of independent attributes to predict the value of a dependent attribute or **target**. Unsupervised learning does not distinguish between dependent and independent attributes. Supervised functions are predictive. Unsupervised functions are descriptive.

> **Note**
>
> In OML4SQL terminology, a **function** is a general type of problem to be solved by a given approach to machine learning. In SQL language terminology, a **function** is an operation that returns a result.
>
> In OML4SQL documentation, the term **function**, or **machine learning function** refers to an OML4SQL machine learning function; the term **SQL function** or **SQL machine learning function** refers to a SQL function for scoring (applying machine learning models).

You can specify any of the values in the following table for the *`mining_function`* parameter to the `CREATE_MODEL` and `CREATE_MODEL2` procedure.

Table 4-2 Oracle Machine Learning mining_function Values

mining_function Value	Description
ASSOCIATION	Association is a descriptive machine learning function. An association model identifies relationships and the probability of their occurrence within a data set (association rules). Association models use the Apriori algorithm.
ATTRIBUTE_IMPORTANCE	Attribute importance is a predictive machine learning function. An attribute importance model identifies the relative importance of attributes in predicting a given outcome. Attribute importance models use the Minimum Description Length algorithm and CUR Matrix Decomposition.
CLASSIFICATION	Classification is a predictive machine learning function. A classification model uses historical data to predict a categorical target. Classification models can use Naive Bayes, Neural Network, Decision Tree, logistic regression, Random Forest, Support Vector Machine, Explicit Semantic Analysis, or XGBoost. The default is Naive Bayes. You can also specify the classification machine learning function for anomaly detection for a One-Class SVM model and a Multivariate State Estimation Technique - Sequential Probability Ratio Test model.
CLUSTERING	Clustering is a descriptive machine learning function. A clustering model identifies natural groupings within a data set. Clustering models can use *k*-Means, O-Cluster, or Expectation Maximization. The default is *k*-Means.
FEATURE_EXTRACTION	Feature extraction is a descriptive machine learning function. A feature extraction model creates a set of optimized attributes. Feature extraction models can use Non-Negative Matrix Factorization, Singular Value Decomposition (which can also be used for Principal Component Analysis) or Explicit Semantic Analysis. The default is Non-Negative Matrix Factorization.
REGRESSION	Regression is a predictive machine learning function. A regression model uses historical data to predict a numerical target. Regression models can use Support Vector Machine, GLM regression, or XGBoost. The default is Support Vector Machine.
TIME_SERIES	Time series is a predictive machine learning function. A time series model forecasts the future values of a time-ordered series of historical numeric data over a user-specified time window. Time series models use the Exponential Smoothing algorithm. The default is Exponential Smoothing.

Related Topics

- Machine Learning Techniques

4.3 Choose the Algorithm

Learn about providing the algorithm settings for a model.

The ALGO_NAME setting specifies the algorithm for a model. If you use the default algorithm for the machine learning technique, or if there is only one algorithm available for the machine learning technique, then you do not need to specify the ALGO_NAME setting.

Table 4-3 Oracle Machine Learning Algorithms

ALGO_NAME Value	Algorithm	Default?	Machine Learning Model Function
ALGO_AI_MDL	Minimum Description Length	—	Attribute importance
ALGO_APRIORI_ASSOCIATION_RULES	Apriori	—	Association
ALGO_CUR_DECOMPOSITION	CUR Matrix Decomposition	—	Attribute importance
ALGO_DECISION_TREE	Decision Tree	—	Classification
ALGO_EXPECTATION_MAXIMIZATION	Expectation Maximization	—	Clustering and Anomaly Detection
ALGO_EXPLICIT_SEMANTIC_ANALYS	Explicit Semantic Analysis	—	Feature extraction and classification
ALGO_EXPONENTIAL_SMOOTHING	Exponential Smoothing	—	Time series and time series regression
ALGO_EXTENSIBLE_LANG	Language used for an extensible algorithm	—	All machine learning functions are supported
ALGO_GENERALIZED_LINEAR_MODEL	Generalized Linear Model	—	Classification and regression
ALGO_KMEANS	*k*-Means	yes	Clustering
ALGO_MSET_SPRT	Multivariate State Estimation Technique - Sequential Probability Ratio Test	—	Anomaly detection (classification with no target)
ALGO_NAIVE_BAYES	Naive Bayes	yes	Classification
ALGO_NEURAL_NETWORK	Neural Network	—	Classification
ALGO_NONNEGATIVE_MATRIX_FACTOR	Non-Negative Matrix Factorization	yes	Feature extraction
ALGO_O_CLUSTER	O-Cluster	—	Clustering
ALGO_RANDOM_FOREST	Random Forest	—	Classification
ALGO_SINGULAR_VALUE_DECOMP	Singular Value Decomposition (can also be used for Principal Component Analysis)	—	Feature extraction
ALGO_SUPPORT_VECTOR_MACHINES	Support Vector Machine	yes	Default regression algorithm; regression, classification, and anomaly detection (classification with no target)
ALGO_XGBOOST	XGBoost	—	Classification and regression

Related Topics

- Specify Model Settings
  You can configure your model by specifying model settings.
- Part III Algorithms

4.4 Automatic Data Preparation

Most algorithms require some form of data transformation. During the model build process, Oracle Machine Learning for SQL can automatically perform the transformations required by the algorithm.

You can choose to supplement the automatic transformations with additional transformations of your own, or you can choose to manage all the transformations yourself.

In calculating automatic transformations, OML4SQL uses heuristics that address the common requirements of a given algorithm. This process results in reasonable model quality in most cases.

Binning and normalization are transformations that are commonly needed by machine learning algorithms.

- Binning
  Binning, also called discretization, is a technique for reducing the cardinality of continuous and discrete data. Binning groups related values together in bins to reduce the number of distinct values.
- Normalization
  Learn about normalization.
- How ADP Transforms the Data
  The following table shows how ADP prepares the data for each algorithm.

Related Topics

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

4.4.1 Binning

Binning, also called discretization, is a technique for reducing the cardinality of continuous and discrete data. Binning groups related values together in bins to reduce the number of distinct values.

Binning can improve resource utilization and model build response time dramatically without significant loss in model quality. Binning can improve model quality by strengthening the relationship between attributes.

Supervised binning is a form of intelligent binning in which important characteristics of the data are used to determine the bin boundaries. In supervised binning, the bin boundaries are identified by a single-predictor decision tree that takes into account the joint distribution with the target. Supervised binning can be used for both numerical and categorical attributes.

4.4.2 Normalization

Learn about normalization.

Normalization is the most common technique for reducing the range of numerical data. Most normalization methods map the range of a single variable to another range (often 0,1).

4.4.3 How ADP Transforms the Data

The following table shows how ADP prepares the data for each algorithm.

Table 4-4 Oracle Machine Learning Algorithms With ADP

Algorithm	Machine Learning Function	Treatment by ADP
Apriori	Association rules	ADP has no effect on association rules.
CUR Matrix Decomposition	Feature selection	ADP has no effect on CUR Matrix Decomposition

Table 4-4 (Cont.) Oracle Machine Learning Algorithms With ADP

Algorithm	Machine Learning Function	Treatment by ADP
Decision Tree	Classification	ADP has no effect on Decision Tree. Data preparation is handled by the algorithm.
Expectation Maximization	Clustering	Single-column (not nested) numerical columns that are modeled with Gaussian distributions are normalized. ADP has no effect on the other types of columns.
GLM	Classification and regression	Numerical attributes are normalized.
k-Means	Clustering	Numerical attributes are normalized.
MDL	Attribute importance	All attributes are binned with supervised binning.
MSET-SPRT	Classification (for anomaly detection)	Z-score normalization is performed.
Naive Bayes	Classification	All attributes are binned with supervised binning.
Neural Network	Classification and regression	Numerical attributes are normalized.
NMF	Feature extraction	Numerical attributes are normalized.
O-Cluster	Clustering	Numerical attributes are binned with a specialized form of equi-width binning, which computes the number of bins per attribute automatically. Numerical columns with all nulls or a single value are removed.
Random Forest	Classification	ADP has no effect on Random Forest. Data preparation is handled by the algorithm.
SVD	Feature extraction	Numeric attributes are centered if PCA is selected.
SVM	Classification, anomaly detection, and regression	Numerical attributes are normalized.
XG Boost	Classification and regression	ADP has no effect on XG Boost.

See Also

- *Oracle Database PL/SQL Packages and Types Reference*
- Part III, Algorithms, in *Oracle Machine Learning for SQL Concepts* for more information about algorithm-specific data preparation

4.5 Embed Transformations in a Model

You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

The transformation instructions are embedded in the model and reapplied whenever the model is applied to new data.

The schema of how you can use `xform_list` to embed your transformations is shown here with `CREATE_MODEL` procedure.

```
DBMS_DATA_MINING.CREATE_MODEL2 (
model_name          IN VARCHAR2,
```

```
mining_function       IN VARCHAR2,
data_query          IN CLOB,
set_list            IN SETTING_LIST,
case_id_column_name      IN VARCHAR2 DEFAULT NULL,
target_column_name       IN VARCHAR2 DEFAULT NULL,
xform_list          IN TRANSFORM_LIST DEFAULT NULL);



DBMS_DATA_MINING.CREATE_MODEL(
                  model_name           IN VARCHAR2,
                  mining_function      IN VARCHAR2,
                  data_table_name      IN VARCHAR2,
                  case_id_column_name  IN VARCHAR2,
                  target_column_name   IN VARCHAR2 DEFAULT NULL,
                  settings_table_name  IN VARCHAR2 DEFAULT NULL,
                  data_schema_name     IN VARCHAR2 DEFAULT NULL,
                  settings_schema_name IN VARCHAR2 DEFAULT NULL,
                  xform_list           IN TRANSFORM_LIST DEFAULT NULL);
```

The following examples show how to create an embedded transform list with `CREATE_MODEL` and `CREATE_MODEL2` procedures.

Here is an example with `DBMS_DATA_MINING.CREATE_MODEL` procedure:

```
BEGIN
DBMS_DATA_MINING.DROP_MODEL('model_sample2');
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
CREATE TABLE sett_table (SETTING_NAME  VARCHAR2(30),
                                  SETTING_VALUE VARCHAR2(4000));

BEGIN
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
('KMNS_DISTANCE','KMNS_EUCLIDEAN');
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
('PREP_AUTO','ON');
   INSERT INTO sett_table (SETTING_NAME, SETTING_VALUE) VALUES
('KMNS_DETAILS', 'KMNS_DETAILS_ALL');
END;
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'N_TRANS_ATM', null,
'TO_CHAR(N_TRANS_ATM)', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'BANK_FUNDS', null,
'BANK_FUNDS+BANK_FUNDS+BANK_FUNDS', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'AGE', null,
'log(10,AGE+1)', 'power(10, AGE)-1');

 DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'model_sample2',
    mining_function     => dbms_data_mining.clustering,
    data_table_name     => 'INSUR_CUST_LTV',
```

```
    case_id_column_name => 'customer_id',
    settings_table_name => 'sett_table',
    xform_list          => xformlist);
END;
```

The following example shows how to create an embedded transformation using the `DBMS_DATA_MINING.CREATE_MODEL2` procedure:

```
DECLARE
  xformlist dbms_data_mining_transform.TRANSFORM_LIST;
  v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'N_TRANS_ATM', null,
'TO_CHAR(N_TRANS_ATM)', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'BANK_FUNDS', null,
'BANK_FUNDS+BANK_FUNDS+BANK_FUNDS', null);
  dbms_data_mining_transform.SET_TRANSFORM(xformlist, 'AGE', null,
'log(10,AGE+1)', 'power(10, AGE)-1');

  v_setlst('ALGO_NAME') := 'ALGO_KMEANS';

 DBMS_DATA_MINING.CREATE_MODEL2(
    model_name          => 'model_sample3',
    mining_function     => 'CLUSTERING',
    data_query          => 'select * from INSUR_CUST_LTV',
    set_list            => v_setlst,
    case_id_column_name => 'customer_id',
    xform_list          => xformlist);
END;
```

- Build a Transformation List
  You can build transformation list by `SET_TRANSFORM`, `STACK`, and `GET_*` methods. These methods are listed here.
- Transformation List and Automatic Data Preparation
  You can provide transformation list and Automatic Data Preparation (ADP) to customize the data transformation.
- Specify Transformation Instructions for an Attribute
  You can pass transformation instructions for an attribute by defining a transformation list.
- Oracle Machine Learning for SQL Transformation Routines
  Learn about transformation routines.
- Understand Reverse Transformations
  Reverse transformations ensure that information returned by the model is expressed in a format that is similar to or the same as the format of the data that was used to train the model. Internal transformation are reversed in the model details and in the results of scoring.

4.5.1 Build a Transformation List

You can build transformation list by SET_TRANSFORM, STACK, and GET_* methods. These methods are listed here.

A transformation list is a collection of transformation records. When a new transformation record is added, it is appended to the top of the transformation list. You can use any of the following methods to build a transformation list:

- The SET_TRANFORM procedure in DBMS_DATA_MINING_TRANSFORM
- The STACK interface in DBMS_DATA_MINING_TRANSFORM
- The GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST functions in DBMS_DATA_MINING
- SET_TRANSFORM
  The SET_TRANSFORM procedure applies a specified SQL expression to a specified attribute.
- The STACK Interface
  The STACK interface creates transformation records from a table of transformation instructions and adds them to a transformation list.
- GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST
  Use the functions to create a new transformation list.

4.5.1.1 SET_TRANSFORM

The SET_TRANSFORM procedure applies a specified SQL expression to a specified attribute.

The SET_TRANSFORM procedure adds a single transformation record to a transformation list.

```
DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM (
          xform_list               IN OUT NOCOPY TRANSFORM_LIST,
          attribute_name           VARCHAR2,
          attribute_subname        VARCHAR2,
          expression               VARCHAR2,
          reverse_expression       VARCHAR2,
          attribute_spec           VARCHAR2 DEFAULT NULL);
```

SQL expressions that you specify with SET_TRANSFORM must fit within a VARCHAR2. To specify a longer expression, you can use the SET_EXPRESSION procedure, which builds an expression by appending rows to a VARCHAR2 array. For example, the following statement appends a transformation instruction for country_id to a list of transformations called my_xforms. The transformation instruction divides country_id by 10 before algorithmic processing begins. The reverse transformation multiplies country_id by 10.

```
dbms_data_mining_transform.SET_TRANSFORM (my_xforms,
   'country_id', NULL, 'country_id/10', 'country_id*10');
```

The reverse transformation is applied in the model details. If country_id is the target of a supervised model, the reverse transformation is also applied to the scored target.

4.5.1.2 The STACK Interface

The STACK interface creates transformation records from a table of transformation instructions and adds them to a transformation list.

The STACK interface offers a set of pre-defined transformations that you can apply to an attribute or to a group of attributes. For example, you can specify supervised binning for all categorical attributes.

The STACK interface specifies that all or some of the attributes of a given type must be transformed in the same way. For example, STACK_BIN_CAT appends binning instructions for categorical attributes to a transformation list. The STACK interface consists of three steps:

1. A CREATE procedure creates a transformation definition table. For example, CREATE_BIN_CAT creates a table to hold categorical binning instructions. The table has columns for storing the name of the attribute, the value of the attribute, and the bin assignment for the value.
2. An INSERT procedure computes the bin boundaries for one or more attributes and populates the definition table. For example, INSERT_BIN_CAT_FREQ performs frequency-based binning on some or all of the categorical attributes in the data source and populates a table created by CREATE_BIN_CAT.
3. A STACK procedure creates transformation records from the information in the definition table and appends the transformation records to a transformation list. For example, STACK_BIN_CAT creates transformation records for the information stored in a categorical binning definition table and appends the transformation records to a transformation list.

4.5.1.3 GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST

Use the functions to create a new transformation list.

These two functions can be used to create a new transformation list from the transformations embedded in an existing model.

The GET_MODEL_TRANSFORMATIONS function returns a list of embedded transformations.

```
DBMS_DATA_MINING.GET_MODEL_TRANSFORMATIONS (
      model_name     IN VARCHAR2)
RETURN DM_TRANSFORMS PIPELINED;
```

GET_MODEL_TRANSFORMATIONS returns a table of dm_transform objects. Each dm_transform has these fields

```
attribute_name       VARCHAR2(4000)
attribute_subname    VARCHAR2(4000)
expression           CLOB
reverse_expression   CLOB
```

The components of a transformation list are transform_rec, not dm_transform. The fields of a transform_rec are described in Table 4-5. You can call GET_MODEL_TRANSFORMATIONS to convert a list of dm_transform objects to transform_rec objects and append each transform_rec to a transformation list.

```
DBMS_DATA_MINING.GET_TRANSFORM_LIST (
      xform_list           OUT NOCOPY TRANSFORM_LIST,
      model_xforms         IN  DM_TRANSFORMS);
```

> **See Also**
>
> "DBMS_DATA_MINING_TRANSFORM Operational Notes", "SET_TRANSFORM Procedure", "CREATE_MODEL Procedure", and "GET_MODEL_TRANSFORMATIONS Function" in *Oracle Database PL/SQL Packages and Types Reference*

4.5.2 Transformation List and Automatic Data Preparation

You can provide transformation list and Automatic Data Preparation (ADP) to customize the data transformation.

The transformation list argument to `CREATE_MODEL2` and `CREATE_MODEL` interacts with the `PREP_AUTO` setting, which controls ADP:

- When ADP is on and you specify a transformation list, your transformations are applied with the automatic transformations and embedded in the model. The transformations that you specify are processed before the automatic transformations.
- When ADP is off and you specify a transformation list, your transformations are applied and embedded in the model, but no system-generated transformations are performed.
- When ADP is on and you do not specify a transformation list, the system-generated transformations are applied and embedded in the model.
- When ADP is off and you do not specify a transformation list, no transformations are embedded in the model; you must separately prepare the data sets you use for building, testing, and scoring the model.

Related Topics

- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.
- *Oracle Database PL/SQL Packages and Types Reference*

4.5.3 Specify Transformation Instructions for an Attribute

You can pass transformation instructions for an attribute by defining a transformation list.

A transformation list is defined as a table of transformation records. Each record (`transform_rec`) specifies the transformation instructions for an attribute.

```
TYPE transform_rec IS RECORD (
    attribute_name      VARCHAR2(30),
    attribute_subname   VARCHAR2(4000),
    expression          EXPRESSION_REC,
    reverse_expression  EXPRESSION_REC,
    attribute_spec      VARCHAR2(4000));
```

The fields in a transformation record are described in this table.

Table 4-5 Fields in a Transformation Record for an Attribute

Field	Description
`attribute_name` and `attribute_subname`	These fields identify the attribute, as described in "Scoping of Model Attribute Name"
`expression`	A SQL expression for transforming the attribute. For example, this expression transforms the age attribute into two categories: child and adult:[0,19) for 'child' and [19,) for adult `CASE WHEN age < 19 THEN 'child' ELSE 'adult'` Expression and reverse expressions are stored in `expression_rec` objects. See "Expression Records" for details.
`reverse_expression`	A SQL expression for reversing the transformation. For example, this expression reverses the transformation of the age attribute: `DECODE(age,'child','(-Inf,19)','[19,Inf)')`
`attribute_spec`	Specifies special treatment for the attribute. The `attribute_spec` field can be null or it can have one or more of these values: • `FORCE_IN` — For GLM, forces the inclusion of the attribute in the model build when the `ftr_selection_enable` setting is enabled. (`ftr_selection_enable` is disabled by default.) If the model is not using GLM, this value has no effect. `FORCE_IN` cannot be specified for nested attributes or text. • `NOPREP` — When ADP is on, prevents automatic transformation of the attribute. If ADP is not on, this value has no effect. You can specify `NOPREP` for a nested attribute, but not for an individual subname (row) in the nested attribute. • `TEXT` — Indicates that the attribute contains unstructured text. ADP has no effect on this setting. `TEXT` may optionally include subsettings `POLICY_NAME`, `TOKEN_TYPE`, and `MAX_FEATURES`. See Example 4-1 and Example 4-2.

- Expression Records
  Example of a transformation record.
- Attribute Specifications
  Learn how to define the characteristics specific to an attribute through attribute specification.

Related Topics

- Scoping of Model Attribute Name
  Learn about model attribute name.
- Expression Records
  Example of a transformation record.

4.5.3.1 Expression Records

Example of a transformation record.

The transformation expressions in a transformation record are `expression_rec` objects.

```
TYPE expression_rec IS RECORD (
     lstmt       DBMS_SQL.VARCHAR2A,
     lb          BINARY_INTEGER DEFAULT 1,
```

```
    ub          BINARY_INTEGER DEFAULT 0);

TYPE varchar2a IS TABLE OF VARCHAR2(32767)
INDEX BY BINARY_INTEGER;
```

The `lstmt` field stores a `VARCHAR2A`, which allows transformation expressions to be very long, as they can be broken up across multiple rows of `VARCHAR2`. Use the `DBMS_DATA_MINING_TRANSFORM.SET_EXPRESSION` procedure to create an `expression_rec`.

4.5.3.2 Attribute Specifications

Learn how to define the characteristics specific to an attribute through attribute specification.

The attribute specification in a transformation record defines characteristics that are specific to this attribute. If not null, the attribute specification can include values `FORCE_IN`, `NOPREP`, or `TEXT`, as described in Table 4-5.

Example 4-1 An Attribute Specification with Multiple Keywords

If more than one attribute specification keyword is applicable, you can provide them in a comma-delimited list. The following expression is the specification for an attribute in a GLM model. Assuming that the `ftr_selection_enable` setting is enabled, this expression forces the attribute to be included in the model. If ADP is on, automatic transformation of the attribute is not performed.

```
"FORCE_IN,NOPREP"
```

Example 4-2 A Text Attribute Specification

For text attributes, you can optionally specify subsettings `POLICY_NAME`, `TOKEN_TYPE`, and `MAX_FEATURES`. The subsettings provide configuration information that is specific to text transformation. In this example, the transformation instructions for the text content are defined in a text policy named `my_policy` with token type is `THEME`. The maximum number of extracted features is 3000.

```
"TEXT(POLICY_NAME:my_policy)(TOKEN_TYPE:THEME)(MAX_FEATURES:3000)"
```

Related Topics

- Configure a Text Attribute
  Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.

4.5.4 Oracle Machine Learning for SQL Transformation Routines

Learn about transformation routines.

OML4SQL provides routines that implement various transformation techniques in the `DBMS_DATA_MINING_TRANSFORM` package.

- Binning Routines
  Explains binning techniques in OML4SQL.
- Normalization Routines
  Learn about normalization routines in Oracle Machine Learning for SQL.
- Outlier Treatment
  Understand what you must do to treat outliers.
- Routines for Outlier Treatment
  Understand the transformations used for outlier treatment.

Related Topics

- *Oracle Database SQL Language Reference*

4.5.4.1 Binning Routines

Explains binning techniques in OML4SQL.

A number of factors go into deciding a binning strategy. Having fewer values typically leads to a more compact model and one that builds faster, but it can also lead to some loss in accuracy.

Model quality can improve significantly with well-chosen bin boundaries. For example, an appropriate way to bin ages is to separate them into groups of interest, such as children 0-13, teenagers 13-19, youth 19-24, working adults 24-35, and so on.

The following table lists the binning techniques provided by OML4SQL:

Table 4-6 Binning Methods in DBMS_DATA_MINING_TRANSFORM

Binning Method	Description
Top-N Most Frequent Items	You can use this technique to bin categorical attributes. You specify the number of bins. The value that occurs most frequently is labeled as the first bin, the value that appears with the next frequency is labeled as the second bin, and so on. All remaining values are in an additional bin.
Supervised Binning	Supervised binning is a form of intelligent binning, where bin boundaries are derived from important characteristics of the data. Supervised binning builds a single-predictor decision tree to find the interesting bin boundaries with respect to a target. It can be used for numerical or categorical attributes.
Equi-Width Binning	You can use equi-width binning for numerical attributes. The range of values is computed by subtracting the minimum value from the maximum value, then the range of values is divided into equal intervals. You can specify the number of bins or it can be calculated automatically. Equi-width binning must usually be used with outlier treatment.
Quantile Binning	Quantile binning is a numerical binning technique. Quantiles are computed using the SQL analytic function `NTILE`. The bin boundaries are based on the minimum values for each quantile. Bins with equal left and right boundaries are collapsed, possibly resulting in fewer bins than requested.

Related Topics

- Routines for Outlier Treatment
  Understand the transformations used for outlier treatment.

4.5.4.2 Normalization Routines

Learn about normalization routines in Oracle Machine Learning for SQL.

Most normalization methods map the range of a single attribute to another range, typically 0 to 1 or -1 to +1.

Normalization is very sensitive to outliers. Without outlier treatment, most values are mapped to a tiny range, resulting in a significant loss of information.

Table 4-7 Normalization Methods in DBMS_DATA_MINING_TRANSFORM

Transformation	Description
Min-Max Normalization	This technique computes the normalization of an attribute using the minimum and maximum values. The shift is the minimum value, and the scale is the difference between the maximum and minimum values.
Scale Normalization	This normalization technique also uses the minimum and maximum values. For scale normalization, shift = 0, and scale = max{abs(max), abs(min)}.
Z-Score Normalization	This technique computes the normalization of an attribute using the mean and the standard deviation. Shift is the mean, and scale is the standard deviation.

Related Topics

- Routines for Outlier Treatment
  Understand the transformations used for outlier treatment.

4.5.4.3 Outlier Treatment

Understand what you must do to treat outliers.

A value is considered an outlier if it deviates significantly from most other values in the column. The presence of outliers can have a skewing effect on the data and can interfere with the effectiveness of transformations such as normalization or binning.

Outlier treatment methods such as trimming or clipping can be implemented to minimize the effect of outliers.

Outliers represent problematic data, for example, a bad reading due to the unusual condition of an instrument. However, in some cases, especially in the business arena, outliers are perfectly valid. For example, in census data, the earnings for some of the richest individuals can vary significantly from the general population. Do not treat this information as an outlier, since it is an important part of the data. You need domain knowledge to determine outlier handling.

4.5.4.4 Routines for Outlier Treatment

Understand the transformations used for outlier treatment.

Outliers are extreme values, typically several standard deviations from the mean. To minimize the effect of outliers, you can Winsorize or trim the data.

Winsorizing involves setting the tail values of an attribute to some specified value. For example, for a 90% Winsorization, the bottom 5% of values are set equal to the minimum value in the 5th percentile, while the upper 5% of values are set equal to the maximum value in the 95th percentile.

Trimming sets the tail values to NULL. The algorithm treats them as missing values.

Outliers affect the different algorithms in different ways. In general, outliers cause distortion with equi-width binning and min-max normalization.

Table 4-8 Outlier Treatment Methods in DBMS_DATA_MINING_TRANSFORM

Transformation	Description
Trimming	This technique trims the outliers in numeric columns by sorting the non-null values, computing the tail values based on some fraction, and replacing the tail values with nulls.
Windsorizing	This technique trims the outliers in numeric columns by sorting the non-null values, computing the tail values based on some fraction, and replacing the tail values with some specified value.

4.5.5 Understand Reverse Transformations

Reverse transformations ensure that information returned by the model is expressed in a format that is similar to or the same as the format of the data that was used to train the model. Internal transformation are reversed in the model details and in the results of scoring.

Some of the attributes used by the model correspond to columns in the build data. However, because of logic specific to the algorithm, nested data, and transformations, some attributes do not correspond to columns.

For example, a nested column in the training data is not interpreted as an attribute by the model. During the model build,OML4SQL explodes nested columns, and each row (an attribute name/value pair) becomes an attribute.

Some algorithms, for example Support Vector Machine (SVM) and Generalized Linear Model (GLM), only operate on numeric attributes. Any non-numeric column in the build data is exploded into binary attributes, one for each distinct value in the column (SVM). GLM does not generate a new attribute for the most frequent value in the original column. These binary attributes are set to one only if the column value for the case is equal to the value associated with the binary attribute.

Algorithms that generate coefficients present challenges in interpreting the results. Examples are SVM and Non-Negative Matrix Factorization (NMF). These algorithms produce coefficients that are used in combination with the transformed attributes. The coefficients are relevant to the data on the transformed scale, not the original data scale.

For all these reasons, the attributes listed in the model details do not resemble the columns of data used to train the model. However, attributes that undergo embedded transformations, whether initiated by Automatic Data Preparation (ADP) or by a user-specified transformation list, appear in the model details in their pre-transformed state, as close as possible to the original column values. Although the attributes are transformed when they are used by the model, they are visible in the model details in a form that can be interpreted by a user.

Related Topics

- ALTER_REVERSE_EXPRESSION Procedure
- GET_MODEL_TRANSFORMATIONS Function
- Model Detail Views

4.6 The CREATE_MODEL2 Procedure

The CREATE_MODEL2 procedure of the DBMS_DATA_MINING package is a procedure for defining model settings to build a model.

By using the CREATE_MODEL2 procedure, the user does not need to create transient database objects. The model can use configuration settings and user-specified transformations. In the CREATE_MODEL2 procedure, the input is a table or a view and if such an object is not already present, the user must create it.

```
DBMS_DATA_MINING.CREATE_MODEL2 (
model_name            IN VARCHAR2,
mining_function       IN VARCHAR2,
data_query            IN CLOB,
set_list              IN SETTING_LIST,
case_id_column_name   IN VARCHAR2 DEFAULT NULL,
target_column_name    IN VARCHAR2 DEFAULT NULL,
xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

The data_query parameter species a query which provides training data for building the model. The set_list parameter specifies the SETTING_LIST. SETTING_LIST is a table of CLOB index by VARCHAR2(30); Where the index is the setting name and the CLOB is the setting value for that name. The rest of the parameters are covered in the CREATE_MODEL procedure.

You can also rename the model using the RENAME_MODEL procedure of the DBMS_DATA_MINING package. The procedure changes the value of the machine learning model specified against MODEL_NAME with another name that you specify.

The following CREATE_MODEL2 procedure builds a classification model using SVM algorithm. The following example mining_data_build_v data set to arrive at likelihood of customers opting the affinity card program. .

```
DECLARE
    v_setlist DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlist('PREP_AUTO') := 'ON';
    v_setlist('ALGO_NAME') := 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlist('SVMS_KERNEL_FUNCTION') := 'SVMS_LINEAR';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MODEL',
        MINING_FUNCTION     => 'CLASSIFICATION',
        DATA_QUERY          => 'select * from mining_data_build_v',
        SET_LIST            => v_setlist,
        CASE_ID_COLUMN_NAME => 'CUST_ID',
    TARGET_COLUMN_NAME  => 'AFFINITY_CARD');
END;
```

Related Topics

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

4.7 The CREATE_MODEL Procedure

The CREATE_MODEL procedure of the DBMS_DATA_MINING package uses the specified data to create a machine learning model with the specified name and machine learning function.

The model can be created with configuration settings and user-specified transformations.

```
PROCEDURE CREATE_MODEL(
                  model_name            IN VARCHAR2,
                  mining_function       IN VARCHAR2,
                  data_table_name       IN VARCHAR2,
                  case_id_column_name   IN VARCHAR2,
                  target_column_name    IN VARCHAR2 DEFAULT NULL,
                  settings_table_name   IN VARCHAR2 DEFAULT NULL,
                  data_schema_name      IN VARCHAR2 DEFAULT NULL,
                  settings_schema_name  IN VARCHAR2 DEFAULT NULL,
                  xform_list            IN TRANSFORM_LIST DEFAULT NULL);
```

You can also rename the model using the RENAME_MODEL procedure of the DBMS_DATA_MINING package. The procedure changes the value of the machine learning model specified against MODEL_NAME with another name that you specify.

The following example builds a classification model using the Support Vector Machine algorithm.

```
 Create the settings table
CREATE TABLE svm_model_settings (
  setting_name  VARCHAR2(30),
  setting_value VARCHAR2(30));

-- Populate the settings table
-- Specify SVM. By default, Naive Bayes is used for classification.
-- Specify ADP. By default, ADP is not used.
BEGIN
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
     (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
  COMMIT;
END;
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svm_model_settings');
END;
/
```

Related Topics

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

4.8 Specify Model Settings

You can configure your model by specifying model settings.

Numerous configuration settings are available for configuring machine learning models at build time. Specify your model settings in `CREATE_MODEL` or `CREATE_MODEL2` procedures. To specify settings in `CREATE_MODEL` procedure, create a settings table with the columns shown in the following table and pass the table to in the procedure.

You can also use `CREATE_MODEL2` procedure where you can directly pass the model settings to a variable that can be used in the procedure. The variable can be declared with `DBMS_DATA_MINING.SETTING_LIST` procedure.

Table 4-9 Settings Table Required Columns

Column Name	Data Type
`setting_name`	`VARCHAR2(30)`
`setting_value`	`VARCHAR2(4000)`

Example 4-3 creates a settings table for a Support Vector Machine (SVM) classification model. Since SVM is not the default classifier, the `ALGO_NAME` setting is used to specify the algorithm. Setting the `SVMS_KERNEL_FUNCTION` to `SVMS_LINEAR` causes the model to be built with a linear kernel. If you do not specify the kernel function, the algorithm chooses the kernel based on the number of attributes in the data.

Example 4-4 creates a model with the model settings that are stored in a variable from `SETTING_LIST`.

Some settings apply generally to the model, others are specific to an algorithm. Model settings are referenced in Table 4-10 and Table 4-11.

Table 4-10 General Model Settings

Settings	Description
Machine learning function settings	Machine Learning Technique Settings
Algorithm names	Algorithm Names
Global model characteristics	Global Settings
Automatic Data Preparation	Automatic Data Preparation

Table 4-11 Algorithm-Specific Model Settings

Algorithm	Description
CUR Matrix Decomposition	DBMS_DATA_MINING —Algorithm Settings: CUR Matrix Decomposition
Decision Tree	DBMS_DATA_MINING —Algorithm Settings: Decision Tree
Expectation Maximization	DBMS_DATA_MINING —Algorithm Settings: Expectation Maximization
Explicit Semantic Analysis	DBMS_DATA_MINING —Algorithm Settings: Explicit Semantic Analysis
Exponential Smoothing	DBMS_DATA_MINING —Algorithm Settings: Exponential Smoothing Models
Generalized Linear Model	DBMS_DATA_MINING —Algorithm Settings: Generalized Linear Models

Table 4-11 (Cont.) Algorithm-Specific Model Settings

Algorithm	Description
k-Means	DBMS_DATA_MINING —Algorithm Settings: *k*-Means
Multivariate State Estimation Technique - Sequential Probability Ratio Test	DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test
Naive Bayes	Algorithm Settings: Naive Bayes
Neural Network	DBMS_DATA_MINING —Algorithm Settings: Neural Network
Non-Negative Matrix Factorization	DBMS_DATA_MINING —Algorithm Settings: Non-Negative Matrix Factorization
O-Cluster	Algorithm Settings: O-Cluster
Random Forest	DBMS_DATA_MINING — Algorithm Settings: Random Forest
Singular Value Decomposition	DBMS_DATA_MINING —Algorithm Settings: Singular Value Decomposition
Support Vector Machine	DBMS_DATA_MINING —Algorithm Settings: Support Vector Machine
XGBoost	DBMS_DATA_MINING — Algorithm Settings: XGBoost

> **Note**
>
> Some XGBoost objectives apply only to classification function models and other objectives apply only to regression function models. If you specify an incompatible `objective` value, an error is raised. In the `DBMS_DATA_MINING.CREATE_MODEL` procedure, if you specify `DBMS_DATA_MINING.CLASSIFICATION` as the function, then the only objective values that you can use are the `binary` and `multi` values. The one exception is `binary: logitraw`, which produces a continuous value and applies only to a regression model. If you specify `DBMS_DATA_MINING.REGRESSION` as the function, then you can specify `binary: logitraw` or any of the `count`, `rank`, `reg`, and `survival` values as the objective.
>
> The values for the XGBoost objective setting are listed in the Settings for Learning Tasks table in DBMS_DATA_MINING — Algorithm Settings: XGBoost.

Example 4-3 Creating a Settings Table and Creating an SVM Classification Model Using CREATE.MODEL procedure

```
CREATE TABLE svmc_sh_sample_settings (
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(4000));

BEGIN
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.svms_kernel_function, dbms_data_mining.svms_linear);
  COMMIT;
END;
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
```

```
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => 'affinity_card',
    settings_table_name => 'svmc_sh_sample_settings');
END;
```

Example 4-4 Specify Model Settings for a SVM Classification Model Using CREATE_MODEL2 procedure

```
DECLARE
    v_setlist DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlist('PREP_AUTO') := 'ON';
    v_setlist('ALGO_NAME') := 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlist('SVMS_KERNEL_FUNCTION') := 'SVMS_LINEAR';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MODEL',
        MINING_FUNCTION     => 'CLASSIFICATION',
        DATA_QUERY          => 'select * from mining_data_build_v',
        SET_LIST            => v_setlist,
        CASE_ID_COLUMN_NAME => 'CUST_ID,
    TARGET_COLUMN_NAME  => 'AFFINITY_CARD');
END;
```

- Specify Costs
  Specify a cost matrix table to build a Decision Tree model.
- Specify Prior Probabilities
  Prior probabilities can be used to offset differences in distribution between the build data and the actual population.
- Specify Class Weights
  Specify class weights table settings in logistic regression or Support Vector Machine (SVM) classification to favor higher weighted classes.
- About Partitioned Models
  Partitioned models allow you to divide your data set into multiple partitions based on specific attributes and build a model for each partition. The system automates the creation and management of these models, reducing manual effort.
- Model Settings in the Data Dictionary
  Explains about `ALL/USER/DBA_MINING_MODEL_SETTINGS` in data dictionary view.
- Specify Oracle Machine Learning Model Settings for an R Model

Related Topics

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

4.8.1 Specify Costs

Specify a cost matrix table to build a Decision Tree model.

The `CLAS_COST_TABLE_NAME` setting specifies the name of a cost matrix table to be used in building a Decision Tree model. A cost matrix biases a classification model to minimize costly misclassifications. The cost matrix table must have the columns shown in the following table:

Table 4-12 Cost Matrix Table Required Columns

Column Name	Data Type
actual_target_value	valid target data type
predicted_target_value	valid target data type
cost	NUMBER

Decision Tree is the only algorithm that supports a cost matrix at build time. However, you can create a cost matrix and associate it with any classification model for scoring.

If you want to use costs for scoring, create a table with the columns shown in Table 4-12, and use the DBMS_DATA_MINING.ADD_COST_MATRIX procedure to add the cost matrix table to the model. You can also specify a cost matrix inline when invoking a PREDICTION function. Table 3-1 has details for valid target data types.

Related Topics

- *Oracle Machine Learning for SQL Concepts*

4.8.2 Specify Prior Probabilities

Prior probabilities can be used to offset differences in distribution between the build data and the actual population.

The CLAS_PRIORS_TABLE_NAME setting specifies the name of a table of prior probabilities to be used in building a Naive Bayes model. The priors table must have the columns shown in the following table.

Table 4-13 Priors Table Required Columns

Column Name	Data Type
target_value	valid target data type
prior_probability	NUMBER

Related Topics

- Target Attribute
  Understand what a **target** means in machine learning and understand the different target data types.
- *Oracle Machine Learning for SQL Concepts*

4.8.3 Specify Class Weights

Specify class weights table settings in logistic regression or Support Vector Machine (SVM) classification to favor higher weighted classes.

The CLAS_WEIGHTS_TABLE_NAME setting specifies the name of a table of class weights to be used to bias a logistic regression (Generalized Linear Model classification) or SVM classification model to favor higher weighted classes. The weights table must have the columns shown in the following table.

Table 4-14 Class Weights Table Required Columns

Column Name	Data Type
target_value	Valid target data type
class_weight	NUMBER

Related Topics

- Target Attribute
  Understand what a **target** means in machine learning and understand the different target data types.
- *Oracle Machine Learning for SQL Concepts*

4.8.6 Specify Oracle Machine Learning Model Settings for an R Model

 This topic applies only to Oracle on-premises.

The machine learning model settings for an R language model determine the characteristics of the model and are specified in the model settings table.

You can build a machine learning model in the R language by specifying R as the value of the ALGO_EXTENSIBLE_LANG setting in the model settings table. You can create a model by combining in the settings table generic settings that do not require an algorithm, such as ODMS_PARTITION_COLUMNS and ODMS_SAMPLING. You can also specify the following settings, which are exclusive to an R machine learning model.

- ALGO_EXTENSIBLE_LANG
  Use the ALGO_EXTENSIBLE_LANG setting to specify the language for the Oracle Machine Learning for SQL extensible algorithm framework.
- RALG_BUILD_FUNCTION
  Use the RALG_BUILD_FUNCTION setting to specify the name of an existing registered R script for building an Oracle Machine Learning for SQL model using the R language.
- RALG_DETAILS_FUNCTION
  The RALG_DETAILS_FUNCTION specifies the R model metadata that is returned in the R data.frame.
- RALG_DETAILS_FORMAT
  Use the RALG_DETAILS_FORMAT setting to specify the names and column types in the model view.
- RALG_SCORE_FUNCTION
  Use the RALG_SCORE_FUNCTION setting to specify an existing registered R script for R algorithm machine learning model to use for scoring data.
- RALG_WEIGHT_FUNCTION
  Use the RALG_WEIGHT_FUNCTION setting to specify the name of an existing registered R script that computes the weight or contribution for each attribute in scoring. The specified R script is used in the SQL function PREDICTION_DETAILS to evaluate attribute contribution.
- Registered R Scripts
  The RALG_*_FUNCTION settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

- Algorithm Metadata Registration
  Algorithm metadata registration allows for a uniform and consistent approach of registering new algorithm functions and their settings.

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

4.8.6.1 ALGO_EXTENSIBLE_LANG

Use the `ALGO_EXTENSIBLE_LANG` setting to specify the language for the Oracle Machine Learning for SQL extensible algorithm framework.

Currently, `R` is the only valid value for the `ALGO_EXTENSIBLE_LANG` setting. When you set the value for `ALGO_EXTENSIBLE_LANG` to R, the machine learning models are built using the R language. You can use the following settings in the settings table to specify the characteristics of the R model.

- RALG_BUILD_FUNCTION
- RALG_BUILD_PARAMETER
- RALG_DETAILS_FUNCTION
- RALG_DETAILS_FORMAT
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

4.8.6.2 RALG_BUILD_FUNCTION

Use the `RALG_BUILD_FUNCTION` setting to specify the name of an existing registered R script for building an Oracle Machine Learning for SQL model using the R language.

You must specify both the `RALG_BUILD_FUNCTION` and `ALGO_EXTENSIBLE_LANG` settings in the model settings table. The R script defines an R function that has as the first input argument an R `data.frame` object for training data. The function returns an Oracle Machine Learning model object. The first data argument is mandatory. The `RALG_BUILD_FUNCTION` can accept additional model build parameters.

> **Note**
>
> The valid inputs for input parameters are numeric and string scalar data types.

Example 4-5 Example of RALG_BUILD_FUNCTION

This example shows how to specify the name of the R script *MY_LM_BUILD_SCRIPT* that is used to build the model.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_build_function,'MY_LM_BUILD_SCRIPT');
End;
/
```

The R script *MY_LM_BUILD_SCRIPT* defines an R function that builds the LM model. You must register the script *MY_LM_BUILD_SCRIPT* in the Oracle Machine Learning for R script repository which uses the existing OML4R security restrictions. You can use the OML4R `sys.rqScriptCreate` procedure to register the script. OML4R requires the `RQADMIN` role to register R scripts.

For example:

```
Begin
sys.rqScriptCreate('MY_LM_BUILD_SCRIPT', 'function(data, formula,
model.frame) {lm(formula = formula, data=data, model =
as.logical(model.frame)}');
End;
/
```

For Clustering and Feature Extraction machine learning function model builds, the R attributes `dm$nclus` and `dm$nfeat` must be set on the return R model to indicate the number of clusters and features respectively.

The R script `MY_KM_BUILD_SCRIPT` defines an R function that builds the *k*-Means model for clustering. The R attribute `dm$nclus` is set with the number of clusters for the returned clustering model.

```
'function(dat) {dat.scaled <- scale(dat)
     set.seed(6543); mod <- list()
     fit <- kmeans(dat.scaled, centers = 3L)
     mod[[1L]] <- fit
     mod[[2L]] <- attr(dat.scaled, "scaled:center")
     mod[[3L]] <- attr(dat.scaled, "scaled:scale")
     attr(mod, "dm$nclus") <- nrow(fit$centers)
     mod}'
```

The R script `MY_PCA_BUILD_SCRIPT` defines an R function that builds the PCA model. The R attribute `dm$nfeat` is set with the number of features for the returned feature extraction model.

```
'function(dat) {
     mod <- prcomp(dat, retx = FALSE)
     attr(mod, "dm$nfeat") <- ncol(mod$rotation)
     mod}'
```

- RALG_BUILD_PARAMETER
  The `RALG_BUILD_FUNCTION` input parameter specifies a list of numeric and string scalar values in SQL `SELECT` query statement format.

Related Topics

- RALG_BUILD_PARAMETER
  The RALG_BUILD_FUNCTION input parameter specifies a list of numeric and string scalar values in SQL SELECT query statement format.
- Registered R Scripts
  The RALG_*_FUNCTION settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

4.8.6.2.1 RALG_BUILD_PARAMETER

The RALG_BUILD_FUNCTION input parameter specifies a list of numeric and string scalar values in SQL SELECT query statement format.

Example 4-6 Example of RALG_BUILD_PARAMETER

The RALG_BUILD_FUNCTION input parameters must be a list of numeric and string scalar values. The input parameters are optional.

The syntax of the parameter is:

```
'SELECT value parameter name ...FROM dual'
```

This example shows how to specify a formula for the input argument 'formula' and a numeric value of zero for input argument 'model.frame' using the RALG_BUILD_PARAMETER. These input arguments must match with the function signature of the R script used in the RALG_BUILD_FUNCTION parameter.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_build_parameter, 'select ''AGE ~ .'' as "formula", 0
as "model.frame" from dual');
End;
/
```

Related Topics

- RALG_BUILD_FUNCTION
  Use the RALG_BUILD_FUNCTION setting to specify the name of an existing registered R script for building an Oracle Machine Learning for SQL model using the R language.

4.8.6.3 RALG_DETAILS_FUNCTION

The RALG_DETAILS_FUNCTION specifies the R model metadata that is returned in the R data.frame.

Use the RALG_DETAILS_FUNCTION to specify an existing registered R script that generates model information. The script defines an R function that contains the first input argument for the R model object. The output of the R function must be a data.frame. The columns of the data.frame are defined by the RALG_DETAILS_FORMAT setting, and may contain only numeric or string scalar types.

Example 4-7 Example of RALG_DETAILS_FUNCTION

This example shows how to specify the name of the R script `MY_LM_DETAILS_SCRIPT` in the model settings table. This script defines the R function that is used to provide the model information.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_details_function, 'MY_LM_DETAILS_SCRIPT');
End;
/
```

In the Oracle Machine Learning for R script repository, the script *`MY_LM_DETAILS_SCRIPT`* is registered as:

```
 'function(mod) data.frame(name=names(mod$coefficients),
    coef=mod$coefficients)'
```

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.
- RALG_DETAILS_FORMAT
  Use the `RALG_DETAILS_FORMAT` setting to specify the names and column types in the model view.

4.8.6.4 RALG_DETAILS_FORMAT

Use the `RALG_DETAILS_FORMAT` setting to specify the names and column types in the model view.

The value of the setting is a string that contains a `SELECT` statement to specify a list of numeric and string scalar data types for the name and type of the model view columns.

When the `RALG_DETAILS_FORMAT` and `RALG_DETAILS_FUNCTION` settings are both specified, a model view by the name `DM$VD` <*`model_name`*> is created along with an R model in the current schema. The first column of the model view is `PARTITION_NAME`. It has the value `NULL` for non-partitioned models. The other columns of the model view are defined by `RALG_DETAILS_FORMAT` setting.

Example 4-8 Example of RALG_DETAILS_FORMAT

This example shows how to specify the name and type of the columns for the generated model view. The model view contains the `varchar2` column `attr_name` and the number column `coef_value` after the first column `partition_name`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_details_format, 'select cast(''a'' as varchar2(20)) as
attr_name, 0 as coef_value from dual');
End;
/
```

Related Topics

- RALG_DETAILS_FUNCTION
  The `RALG_DETAILS_FUNCTION` specifies the R model metadata that is returned in the R `data.frame`.

4.8.6.5 RALG_SCORE_FUNCTION

Use the `RALG_SCORE_FUNCTION` setting to specify an existing registered R script for R algorithm machine learning model to use for scoring data.

The specified R script defines an R function. The first input argument defines the model object. The second input argument defines the R `data.frame` that is used for scoring data.

Example 4-9 Example of RALG_SCORE_FUNCTION

This example shows how the R function takes the Linear Model model and scores the data in the `data.frame`. The function argument `object` is the LM model. The argument `newdata` is a `data.frame` containing the data to score.

```
function(object, newdata) {res <- predict.lm(object, newdata = newdata,
se.fit = TRUE); data.frame(fit=res$fit, se=res$se.fit,
df=summary(object)$df[1L])}
```

The output of the R function must be a `data.frame`. Each row represents the prediction for the corresponding scoring data from the input `data.frame`. The columns of the `data.frame` are specific to machine learning functions, such as:

Regression: A single numeric column for the predicted target value, with two optional columns containing the standard error of the model fit, and the degrees of freedom number. The optional columns are needed for the SQL function `PREDICTION_BOUNDS` to work.

Example 4-10 Example of RALG_SCORE_FUNCTION for Regression

This example shows how to specify the name of the R script *`MY_LM_PREDICT_SCRIPT`* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_LM_PREDICT_SCRIPT');
End;
/
```

In the Oracle Machine Learning for R script repository, the script *`MY_LM_PREDICT_SCRIPT`* is registered as:

```
function(object, newdata) {data.frame(pre = predict(object, newdata =
newdata))}
```

Classification: Each column represents the predicted probability of one target class. The column name is the target class name.

Example 4-11 Example of RALG_SCORE_FUNCTION for Classification

This example shows how to specify the name of the R script *MY_LOGITGLM_PREDICT_SCRIPT* that is used to score the logit Classification model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_LOGITGLM_PREDICT_SCRIPT');
End;
/
```

In the OML4R script repository, *MY_LOGITGLM_PREDICT_SCRIPT* is registered as follows. It is a logit Classification with two target classes, "0" and "1".

```
'function(object, newdata) {
   pred <- predict(object, newdata = newdata, type="response");
   res <- data.frame(1-pred, pred);
   names(res) <- c("0", "1");
   res}'
```

Clustering: Each column represents the predicted probability of one cluster. The columns are arranged in order of cluster ID. Each cluster is assigned a cluster ID, and they are consecutive values starting from 1. To support `CLUSTER_DISTANCE` in the R model, the output of R score function returns an extra column containing the value of the distance to each cluster in order of cluster ID after the columns for the predicted probability.

Example 4-12 Example of RALG_SCORE_FUNCTION for Clustering

This example shows how to specify the name of the R script *MY_CLUSTER_PREDICT_SCRIPT* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_CLUSTER_PREDICT_SCRIPT');
End;
/
```

In the OML4R script repository, the script *MY_CLUSTER_PREDICT_SCRIPT* is registered as:

```
'function(object, dat){
     mod <- object[[1L]]; ce <- object[[2L]]; sc <- object[[3L]];
     newdata = scale(dat, center = ce, scale = sc);
     centers <- mod$centers;
     ss <- sapply(as.data.frame(t(centers)),
     function(v) rowSums(scale(newdata, center=v, scale=FALSE)^2));
     if (!is.matrix(ss)) ss <- matrix(ss, ncol=length(ss));
     disp <- -1 / (2* mod$tot.withinss/length(mod$cluster));
     distr <- exp(disp*ss);
     prob <- distr / rowSums(distr);
     as.data.frame(cbind(prob, sqrt(ss)))}'
```

Feature Extraction: Each column represents the coefficient value of one feature. The columns are arranged in order of feature ID. Each feature is assigned a feature ID, which are consecutive values starting from 1.

Example 4-13 Example of RALG_SCORE_FUNCTION for Feature Extraction

This example shows how to specify the name of the R script *MY_FEATURE_EXTRACTION_SCRIPT* that is used to score the model in the model settings table `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_score_function, 'MY_FEATURE_EXTRACTION_SCRIPT');
End;
/
```

In the OML4R script repository, the script *MY_FEATURE_EXTRACTION_SCRIPT* is registered as:

```
 'function(object, dat) { as.data.frame(predict(object, dat)) }'
```

The function fetches the centers of the features from the R model, and computes the feature coefficient based on the distance of the score data to the corresponding feature center.

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

4.8.6.6 RALG_WEIGHT_FUNCTION

Use the `RALG_WEIGHT_FUNCTION` setting to specify the name of an existing registered R script that computes the weight or contribution for each attribute in scoring. The specified R script is used in the SQL function `PREDICTION_DETAILS` to evaluate attribute contribution.

The specified R script defines an R function containing the first input argument for a model object, and the second input argument of an R `data.frame` for scoring data. When the machine learning function is Classification, Clustering, or Feature Extraction, the target class name, cluster ID, or feature ID is passed by the third input argument to compute the weight for that particular class, cluster, or feature. The script returns a `data.frame` containing the contributing weight for each attribute in a row. Each row corresponds to that input scoring `data.frame`.

Example 4-14 Example of RALG_WEIGHT_FUNCTION

This example specifies the name of the R script *MY_PREDICT_WEIGHT_SCRIPT* that computes the weight or contribution of R model attributes in the `model_setting_table`.

```
Begin
insert into model_setting_table values
(dbms_data_mining.ralg_weight_function, 'MY_PREDICT_WEIGHT_SCRIPT');
End;
/
```

In the Oracle Machine Learning for R script repository, the script *MY_PREDICT_WEIGHT_SCRIPT* for Regression is registered as:

```
'function(mod, data) { coef(mod)[-1L]*data }'
```

In the OML4R script repository, the script *MY_PREDICT_WEIGHT_SCRIPT* for logit Classification is registered as:

```
'function(mod, dat, clas) {
   v <- predict(mod, newdata=dat, type = "response");
   v0 <- data.frame(v, 1-v); names(v0) <- c("0", "1");
   res <- data.frame(lapply(seq_along(dat),
   function(x, dat) {
   if(is.numeric(dat[[x]])) dat[,x] <- as.numeric(0)
   else dat[,x] <- as.factor(NA);
   vv <- predict(mod, newdata = dat, type = "response");
   vv = data.frame(vv, 1-vv); names(vv) <- c("0", "1");
   v0[[clas]] / vv[[clas]]}, dat = dat));
   names(res) <- names(dat);
   res}'
```

Related Topics

- Registered R Scripts
  The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

4.8.6.7 Registered R Scripts

The `RALG_*_FUNCTION` settings must specify R scripts that exist in the Oracle Machine Learning for R script repository.

You can register the R scripts using the OML4R SQL procedure `sys.rqScriptCreate`. To register a scripts, you must have the `RQADMIN` role.

The `RALG_*_FUNCTION` settings include the following functions:

- RALG_BUILD_FUNCTION
- RALG_DETAILS_FUNCTION
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

> **Note**
>
> The R scripts must exist in the OML4R script repository for an R model to function.

After an R model is built, the name of the specified R script become a model setting. These R script must exist in the OML4R script repository for an R model to remain functional.

You can manage the R memory that is used to build, score, and view the R models through OML4R as well.

4.8.6.8 Algorithm Metadata Registration

Algorithm metadata registration allows for a uniform and consistent approach of registering new algorithm functions and their settings.

User have the ability to add new algorithms through the `REGISTER_ALGORITHM` procedure registration process. The new algorithms can appear as available within Oracle Machine Learning for SQL for their appropriate machine learning functions. Based on the registration metadata, the settings page is dynamically rendered. Algorithm metadata registration extends the machine learning model capability of OML4SQL.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- FETCH_JSON_SCHEMA Procedure
- REGISTER_ALGORITHM Procedure
- JSON Schema for R Extensible Algorithm

4.8.4 About Partitioned Models

Partitioned models allow you to divide your data set into multiple partitions based on specific attributes and build a model for each partition. The system automates the creation and management of these models, reducing manual effort.

When you build a model on a data set, a single generic model may not perform well on new or evolving data. To address this, you can specify partition columns to create separate models for each partition based on some characteristics. For example, if your data set includes a "REGION" attribute with four values, four models will be created automatically, each corresponding to a region. The system manages these sub-models as part of a single partitioned model.

The partitioned model resides as a first-class, persistent database object, with all sub-models stored on disk. This structure improves performance, especially for models with a large number of partitions, and allows for efficient management, such as dropping individual sub-models when needed.

When building or scoring partitioned models, not all sub-models need to be loaded into memory simultaneously. This approach optimizes memory usage and enhances processing efficiency. The system provides a single model for scoring, while users can still access individual component models if needed.

- Partitioned Model Build Process
  To build a partitioned model, Oracle Machine Learning for SQL requires a partitioning key specified in a settings table.
- DDL in Partitioned model
  Learn about maintenance of partitioned models thorough DDL operations.
- Partitioned Model Scoring
  The scoring of the partitioned model is the same as that of the non-partitioned model.

Related Topics

- *Oracle Database Reference*

> **See Also**
>
> *Oracle Machine Learning for SQL User's Guide* to understand more about partitioned models.

4.8.4.1 Partitioned Model Build Process

To build a partitioned model, Oracle Machine Learning for SQL requires a partitioning key specified in a settings table.

The partitioning key is a comma-separated list of one or more columns (up to 16) from the input data set. The partitioning key horizontally slices the input data based on discrete values of the partitioning key. That is, partitioning is performed as list values as opposed to range partitioning against a continuous value. The partitioning key supports only columns of the data type `NUMBER` and `VARCHAR2`.

During the build process the input data set is partitioned based on the distinct values of the specified key. Each data slice (unique key value) results in its own model partition. The resultant model partition is not separate and is not visible to you as a standalone model. The default value of the maximum number of partitions for partitioned models is 1000 partitions. You can also set a different maximum partitions value. If the number of partitions in the input data set exceeds the defined maximum, OML4SQL throws an exception.

The partitioned model organizes features common to all partitions and the partition specific features. The common features consist of the following metadata:

- The model name
- The machine learning function
- The machine learning algorithm
- A super set of all machine learning model attributes referenced by all partitions (signature)
- A common set of user-defined column transformations
- Any user-specified or default build settings that are interpreted as global; for example, the Auto Data Preparation (ADP) setting

4.8.4.2 DDL in Partitioned model

Learn about maintenance of partitioned models thorough DDL operations.

Partitioned models are maintained through the following DDL operations:

- Add Partition
  Oracle Machine Learning for SQL supports adding a single partition or multiple partitions to an existing partitioned model.
- Drop Partition
  Oracle Machine Learning for SQL supports dropping a single model partition for a given partition name.
- Replace Partition
  Oracle Machine Learning for SQL supports replacing an existing partition for which conflicting keys are found.

- Drop Partitioned Model
  OML4SQL supports dropping a partitioned model. This is used when the full model is obsolete or no longer required.

4.8.4.2.1 Add Partition

Oracle Machine Learning for SQL supports adding a single partition or multiple partitions to an existing partitioned model.

The addition occurs based on the input data set and the name of the existing partitioned model. The operation takes the input data set and the existing partitioned model as parameters. The partition keys are extracted from the input data set and the model partitions are built against the input data set. These partitions are added to the partitioned model.

If the input data set contains multiple keys, then the operation may create one or more partitions. If the total number of partitions in the model increases to more than the user-defined maximum specified when the model was created, then you get an error. The default threshold value for the number of partitions is 1000. See `ODMS_MAX_PARTITIONS` in DBMS_DATA_MINING - Global Settings for more details.

> **Tip**
>
> In the case where partition keys for new partitions conflict with the existing partitions in the model, you can select from the following three approaches to resolve the conflicts:
>
> - `ERROR`: Terminates the ADD operation without adding any partitions.
> - `REPLACE`: Replaces the existing partition for which the conflicting keys are found.
> - `IGNORE`: Eliminates the rows having the conflicting keys.

The `ODMS_PARTITIONED_COLUMNS` parameter is used for selecting the column on which partition is applied. The following example shows selecting the column for partition:

```
%script

BEGIN DBMS_DATA_MINING.DROP_MODEL('SVM_MOD_PARTITIONED');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME')                 := 'ALGO_SUPPORT_VECTOR_MACHINES';
-- SVM algorithm
    v_setlst('SVMS_KERNEL_FUNCTION')      := 'SVMS_LINEAR';      -- choose
linear kernel, which provides coefficients
    v_setlst('ODMS_PARTITION_COLUMNS')    := 'CUST_GENDER';      -- choose
column on which to partition the data
    v_setlst('ODMS_PARTITION_BUILD_TYPE') := 'ODMS_PARTITION_BUILD_INTRA';
-- Each partition is built in parallel using all replicas

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MOD_PARTITIONED',
        MINING_FUNCTION     => 'REGRESSION',
        DATA_QUERY          => 'SELECT * FROM CUSTOMERS_DEMO',
        SET_LIST            => v_setlst,
```

```
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        TARGET_COLUMN_NAME  => 'YRS_RESIDENCE');
END;
```

You can also specify `ODMS_PARTITION_BUILD_TYPE` to control the parallel build of partitioned models. See DBMS_DATA_MINING - Global Settings for valid values.

The following example shows how you can use `ERROR` as an optional value for `ADD_OPTIONS` which terminates the add operation without adding any partitions.

```
BEGIN
    DBMS_DATA_MINING.ADD_PARTITION(
        model_name  => 'SVM_MOD_PARTITIONED',
        data_query  =>'SELECT * FROM CUSTOMERS_DEMO',
        add_options = >'ERROR'
    );
END;
/
```

> **See Also**
>
> DBMS_DATA_MINING - Global Settings and ADD_PARTITION Procedure to specify the settings.

4.8.4.2.2 Drop Partition

Oracle Machine Learning for SQL supports dropping a single model partition for a given partition name.

Drop a partition when a particular segment of data is no longer relevant or has become obsolete.

> **Note**
>
> If only one partition exists, you cannot directly drop it. Instead, add more partitions before dropping it or drop the entire model.

The following example shows dropping a partition for the `CUST_GENDER` column on which partition was applied.

```
--For dropping a partition with the value 'M'
BEGIN
    DBMS_DATA_MINING.DROP_PARTITION(
        model_name  => 'SVM_MOD_PARTITIONED',
        partition_name  =>'M'

    );
END;
/
```

```
--For dropping a partition with the value 'F'

BEGIN
    DBMS_DATA_MINING.DROP_PARTITION(
        model_name  => 'SVM_MOD_PARTITIONED',
        partition_name  =>'F'

    );
END;
/
```

To add and drop partitions, see:

- ADD_PARTITION Procedure
- DROP_PARTITION Procedure

4.8.4.2.3 Replace Partition

Oracle Machine Learning for SQL supports replacing an existing partition for which conflicting keys are found.

You can use `REPLACE` as a value in the optional parameter `add_options` to define how add operation handles when rows in the input data set conflict with existing partitions in the model.

The following example shows how you can use `REPLACE` as an optional value for `ADD_OPTIONS`.

```
BEGIN
    DBMS_DATA_MINING.ADD_PARTITION(
        model_name  => 'SVM_MOD_PARTITIONED',
        data_query  =>'SELECT * FROM CUSTOMERS_DEMO',
        add_options = >'REPLACE'
    );
END;
/
```

> **See Also**
>
> ADD_PARTITION Procedure

4.8.4.2.4 Drop Partitioned Model

OML4SQL supports dropping a partitioned model. This is used when the full model is obsolete or no longer required.

You can directly drop the entire model. Dropping a partitioned model removes all partitions in one operation.

```
BEGIN DBMS_DATA_MINING.DROP_MODEL('SVM_MOD_PARTITIONED');
EXCEPTION WHEN OTHERS THEN NULL;
END;
/
```

> **See Also**
>
> DROP_MODEL Procedure

4.8.4.3 Partitioned Model Scoring

The scoring of the partitioned model is the same as that of the non-partitioned model.

The syntax of the machine learning function remains the same but is extended to provide an optional hint.

For scoring a partitioned model, the signature columns used during the build for the partitioning key must be present in the scoring data set. These columns are combined to form a unique partition key. The unique key is then mapped to a specific underlying model partition, and the identified model partition is used to score that row.

The partitioned objects that are necessary for scoring are loaded on demand during the query execution and are aged out depending on the System Global Area (SGA) memory.

In this example an SVM model is used to predict the number of years a customer resides at their residence but partitioned on customer gender. The model is then used to predict the target. This example highlights the model settings that you can define when you create a partitioned model. The following example is using a view created from the SH schema tables. The `CREATE_MODEL2` procedure is used for creating the model. The partition attribute is `CUST_GENDER`. This attribute has two options *M* and *F*.

```
%script
BEGIN DBMS_DATA_MINING.DROP_MODEL('SVM_MOD_PARTITIONED');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst DBMS_DATA_MINING.SETTING_LIST;
BEGIN
    v_setlst('ALGO_NAME'):= 'ALGO_SUPPORT_VECTOR_MACHINES';
    v_setlst('SVMS_KERNEL_FUNCTION')  :='SVMS_LINEAR';
    v_setlst('ODMS_PARTITION_COLUMNS'):='CUST_GENDER';

    DBMS_DATA_MINING.CREATE_MODEL2(
        MODEL_NAME          => 'SVM_MOD_PARTITIONED',
        MINING_FUNCTION     => 'REGRESSION',
        DATA_QUERY          => 'SELECT * FROM CUSTOMERS_DEMO',
        SET_LIST            => v_setlst,
        CASE_ID_COLUMN_NAME => 'CUST_ID',
        TARGET_COLUMN_NAME  => 'YRS_RESIDENCE');
END;
```

The output is as follows:

```
PL/SQL procedure successfully completed.


---------------------------
```

```
PL/SQL procedure successfully completed.
```

The following code sample shows the prediction.

```
%script

SELECT cust_id, YRS_RESIDENCE,
       ROUND(PREDICTION(SVM_MOD_PARTITIONED USING *),2) pred_YRS_RESIDENCE
FROM CUSTOMERS_DEMO;



CUST_ID   YRS_RESIDENCE   PRED_YRS_RESIDENCE
   100100               4                 4.71
   100200               2                 1.62
   100300               4                 4.66
   100400               6                  5.9
   100500               2                 2.07
   100600               3                 2.74
   100700               6                 5.78
   100800               5                 7.22
   100900               4                 4.88
   101000               7                 6.49
   101100               4                 3.54
   101200               1                 1.46
   101300               4                 4.34
   101400               4                 4.34 ...
```

The following example retrieves the top 10 most influential attribute-value pairs from a partitioned SVM classification model named `SVM_MOD_PARTITIONED`, for the partition where `CUST_GENDER = 'M'`.

```
SELECT target_value class, attribute_name aname, attribute_value aval,
coefficient coeff
FROM (SELECT target_value, attribute_name, attribute_value, coefficient
  FROM DM$VLSVM_MOD_PARTITIONED WHERE partition_name =
  (SELECT ORA_DM_PARTITION_NAME(SVM_MOD_PARTITIONED using 'M' CUST_GENDER)
FROM dual)
  ORDER BY coefficient DESC)
WHERE ROWNUM <= 10;
```

Use Optional Hint for Scoring a Partitioned Model

The optional hint can impact the performance of a query which involves scoring a partitioned model. See GROUPING Hint for more details.

The following example retrieves the top 3 customers per gender based on prediction probability using a partitioned SVM model named `SVM_MOD_PARTITIONED` and using a `GROUPING` hint.

```
column gender format a1
column income format a30
column rnk format 9
SELECT cust_id, cust_gender as gender, rnk, pd FROM
```

```
( SELECT cust_id, cust_gender,
    PREDICTION_DETAILS(/*+ GROUPING */ SVM_MOD_PARTITIONED, 1 USING *) pd,
    rank() over (partition by cust_gender order by
    PREDICTION_PROBABILITY(SVM_MOD_PARTITIONED, 1 USING *) desc, cust_id) rnk
  FROM mining_data_apply_parallel_v)
WHERE rnk <= 3
order by rnk, cust_gender;
```

See the oml4sql-partitioned-models-svm example on GitHub for a complete example.

Related Topics

- *Oracle Database SQL Language Reference*

4.8.5 Model Settings in the Data Dictionary

Explains about `ALL/USER/DBA_MINING_MODEL_SETTINGS` in data dictionary view.

Information about Oracle Machine Learning model settings can be obtained from the data dictionary view `ALL/USER/DBA_MINING_MODEL_SETTINGS`. When used with the `ALL` prefix, this view returns information about the settings for the models accessible to the current user. When used with the `USER` prefix, it returns information about the settings for the models in the user's schema. The `DBA` prefix is only available for DBAs.

The columns of `ALL_MINING_MODEL_SETTINGS` are described as follows and explained in the following table.

```
describe all_mining_model_settings
```

The output is as follows:

```
Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 OWNER                                     NOT NULL VARCHAR2(30)
 MODEL_NAME                                NOT NULL VARCHAR2(30)
 SETTING_NAME                              NOT NULL VARCHAR2(30)
 SETTING_VALUE                                      VARCHAR2(4000)
 SETTING_TYPE                                       VARCHAR2(7)
```

Table 4-15 ALL_MINING_MODEL_SETTINGS

Column	Description
`owner`	Owner of the machine learning model.
`model_name`	Name of the machine learning model.
`setting_name`	Name of the setting.
`setting_value`	Value of the setting.
`setting_type`	`INPUT` if the value is specified by a user. `DEFAULT` if the value is system-generated.

The following query lists the settings for the Support Vector Machine (SVM) classification model SVMC_SH_CLAS_SAMPLE. The `ALGO_NAME`, `CLAS_WEIGHTS_TABLE_NAME`, and `SVMS_KERNEL_FUNCTION` settings are user-specified. These settings have been specified in a

settings table for the model. The SVMC_SH_CLAS_SAMPLE model is created by the `oml4sql-classification-svm.sql` example.

Example 4-15 ALL_MINING_MODEL_SETTINGS

```
COLUMN setting_value FORMAT A35
 SELECT setting_name, setting_value, setting_type
            FROM all_mining_model_settings
            WHERE model_name in 'SVMC_SH_CLAS_SAMPLE';
```

The output is as follows:

```
SETTING_NAME                   SETTING_VALUE                       SETTING
------------------------------ ----------------------------------- -------
SVMS_ACTIVE_LEARNING           SVMS_AL_ENABLE                      DEFAULT
PREP_AUTO                      OFF                                 DEFAULT
SVMS_COMPLEXITY_FACTOR         0.244212                            DEFAULT
SVMS_KERNEL_FUNCTION           SVMS_LINEAR                         INPUT
CLAS_WEIGHTS_TABLE_NAME        svmc_sh_sample_class_wt             INPUT
SVMS_CONV_TOLERANCE            .001                                DEFAULT
ALGO_NAME                      ALGO_SUPPORT_VECTOR_MACHINES        INPUT
```

Related Topics

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

4.9 Model Detail Views

Model detail views are algorithm-specific. Viewing the model detail views will provide you with additional information about the model you created. The names of model detail views begin with DM$. Some model views, such as Global Name-Value Pairs view (`DM$VG`*model_name*), Computed Settings view (`DM$VS`*model_name*), Model Build Alerts view (`DM$VW`*model_name*), and Normalization and Missing Value Handling view (`DM$VN`*model_name*), are shared by all algorithms and are documented separately. Aside from that, classification, clustering, and regression algorithms share some common views. The columns returned by these views may differ between algorithms.

The following are the model views:

- Model Detail Views for Association Rules
  The model detail view `DM$VR`*model_name* contains the generated rules for association models.
- Model Detail View for Frequent Itemsets
  The model detail view `DM$VI`*model_name* contains information about frequent itemsets.
- Model Detail Views for Transactional Itemsets
  The model detail view `DM$VT`*model_name* contains information about the transactional itemsets.
- Model Detail View for Transactional Rule
  The model detail view `DM$VA`*model_name* contains information about transactional rules and transactional itemsets.
- Model Detail Views for Classification Algorithms
  Model detail views for classification algorithms are the target map view and scoring cost view, which are applicable to all classification algorithms.

- Model Detail Views for CUR Matrix Decomposition
  Model detail views for CUR Matrix Decomposition contain information about the scores and ranks of attributes and rows.
- Model Detail Views for Decision Tree
  The model detail views specific to Decision Tree are the hierarchy view, node statistics view, node description view, and the cost matrix view.
- Model Detail Views for Generalized Linear Model
  Model detail views specific to Generalized Linear Model (GLM) such as details and row diagnostics for linear and logistic regression models are discussed.
- Model Detail View for Multivariate State Estimation Technique - Sequential Probability Ratio Test
  The model detail view specific to Multivariate State Estimation Technique - Sequential Probability Ratio Test contains information about Global Name-Value Paris.
- Model Detail Views for Naive Bayes
  The model detail views specific to Naive Bayes are the prior view and result view.
- Model Detail Views for Neural Network
  Model detail views specific to Neural Network contain information about the weights of the neurons: input layer and hidden layers.
- Model Detail Views for Random Forest
  Model detail views specific to Random Forest contain variable importance measures and statistics.
- Model Detail View for Support Vector Machine
  Model detail views specific to Support Vector Machine (SVM) contain linear coefficients and support vector statistics.
- Model Detail Views for XGBoost
  The model detail views specific to XGBoost contain information about Feature Importance view and Global Name-Value Pairs view.
- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).
- Model Detail Views for Expectation Maximization
  Model detail views specific to Expectation Maximization (EM) contain additional information about an EM model. Additional views are available for EM Clustering, but are absent for EM Anomaly.
- Model Detail Views for k-Means
  Model detail views specific to *k*-Means (KM) contain clustering description view (`DM$VG`), and scoring information.
- Model Detail Views for O-Cluster
  Model detail views specific to O-Cluster (OC) contain information about description view, histograms view, and global view.
- Model Detail Views for Explicit Semantic Analysis
  Model detail views specific to Explicit Semantic Analysis (ESA) contain information about attribute statistics and features.
- Model Detail Views for Non-Negative Matrix Factorization
  Model detail views specific to Non-Negative Matrix Factorization (NMF) contain information about the encoding H matrix and H inverse matrix.
- Model Detail Views for Singular Value Decomposition
  Model detail views specific to Singular Value Decomposition (SVD) contain information about the S matrix, right-singular vectors, and left-singular vectors.

- Model Detail Views for Minimum Description Length
  Model detail views specific to Minimum Description Length (MDL) (for calculating attribute importance) contain information about attribute importance models.
- Model Detail Views for Binning
  The binning view DM$VB describes the bin boundaries used in automatic data preparation.
- Model Detail Views for Global Information
  Model detail views for global information contain information about global statistics, alerts, and computed settings.
- Model Detail Views for Normalization and Missing Value Handling
  The Normalization and Missing Value Handling view DM$VN describes the normalization parameters used in Automatic Data Preparation (ADP) and the missing value replacement when a NULL value is encountered. Missing value replacement applies only to the two-dimensional columns and does not apply to the nested columns.
- Model Detail Views for Exponential Smoothing
  Model detail views specific to Exponential Smoothing (ESM) include information about the model output, global information about the model, and views that support time series regression.
- Model Detail Views for Text Features
  The model details view for text features is DM$VX*model_name*.
- Model Detail Views for ONNX Models
  You can view the details of an embedding model using the model detail views. The names of the views begin with DM$V.

4.9.1 Model Detail Views for Association Rules

The model detail view DM$VR*model_name* contains the generated rules for association models.

These are the available model views for Association Rules:

Model Views	Description
DM$VA*model_name*	Association Rules For Transactional Data
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*:	Association Rule Itemsets
DM$VR*model_name*	Association Rules
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Association Rule Itemsets For Transactional Data
DM$VW*model_name*	Model Build Alerts

Depending on the settings of the model, this rule view (DM$VR*model_name*) different sets of columns. Settings ODMS_ITEM_ID_COLUMN_NAME and ODMS_ITEM_VALUE_COLUMN_NAME determine how each item is defined. If ODMS_ITEM_ID_COLUMN_NAME is set, the input format is called transactional input, otherwise, the input format is called 2-Dimensional input. With transactional input, if setting ODMS_ITEM_VALUE_COLUMN_NAME is not set, each item is defined by ITEM_NAME, otherwise, each item is defined by ITEM_NAME and ITEM_VALUE. With 2-Dimensional input, each item is defined by ITEM_NAME, ITEM_SUBNAME and ITEM_VALUE. Setting ASSO_AGGREGATES specifies the columns to aggregate, which is displayed in the view.

> **Note**
>
> Setting ASSO_AGGREGATES is not allowed for 2-dimensional input.

The following shows the views with different settings.

Transactional Input Without ASSO_AGGREGATES Setting

When you sett ITEM_NAME (ODMS_ITEM_ID_COLUMN_NAME) and do not set ITEM_VALUE (ODMS_ITEM_VALUE_COLUMN_NAME), the view contains the following. The consequent item is defined with only the name field. If you also set ITEM_VALUE, the view has the additional column CONSEQUENT_VALUE that specifies the value field.

```
Name                                      Type
 ----------------------------------------- ----------------------------
 PARTITION_NAME                            VARCHAR2(128)
 RULE_ID                                   NUMBER
 RULE_SUPPORT                              NUMBER
 RULE_CONFIDENCE                           NUMBER
 RULE_LIFT                                 NUMBER
 RULE_REVCONFIDENCE                        NUMBER
 ANTECEDENT_SUPPORT                        NUMBER
 NUMBER_OF_ITEMS                           NUMBER
 CONSEQUENT_SUPPORT                        NUMBER
 CONSEQUENT_NAME                           VARCHAR2(4000)
 ANTECEDENT                                SYS.XMLTYPE
```

Table 4-16 Rule View Columns for Transactional Inputs

Column Name	Description
PARTITION_NAME	A partition in a partitioned model to retrieve details.
RULE_ID	The identifier of the rule.
RULE_SUPPORT	The number of transactions that satisfy the rule.
RULE_CONFIDENCE	The likelihood of a transaction satisfying the rule.
RULE_LIFT	The degree of improvement in the prediction over random chance when the rule is satisfied.
RULE_REVCONFIDENCE	The number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs.
ANTECEDENT_SUPPORT	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions.
NUMBER_OF_ITEMS	The total number of attributes referenced in the antecedent and consequent of the rule.
CONSEQUENT_SUPPORT	The ratio of the number of transactions that satisfy the consequent to the total number of transactions.
CONSEQUENT_NAME	The name of the consequent.
CONSEQUENT_VALUE	The value of the consequent. This column is present when Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as numerical or categorical.

Table 4-16 (Cont.) Rule View Columns for Transactional Inputs

Column Name	Description
ANTECEDENT	The antecedent is described as an itemset. At the itemset level, it specifies the number of aggregates, and if not zero, the names of the columns to be aggregated (as well as the mapping to ASSO_AGG*). The itemset contains >= 1 items. • When ODMS_ITEM_VALUE_COLUMN_NAME is not set, each item is defined by item_name. As an example, if the antecedent contains one item B, then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>B</item_name></item></itemset> As another example, if the antecedent contains two items, A and C, then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>A</item_name></item><item><item_name>C</item_name></item></itemset> • When setting ODMS_ITEM_VALUE_COLUMN_NAME is set, each item is defined by item_name and item_value. As an example, if the antecedent contains two items, (name A, value 1) and (name C, value 1), then it is represented as follows: <itemset NUMAGGR="0"><item><item_name>A</item_name><item_value>1</item_value></item><item><item_name>C</item_name><item_value>1</item_value></item></itemset>

Transactional Input With ASSO_AGGREGATES Setting

Similar to the view without an aggregates setting, there are three cases:

- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is not set.
- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as numerical, the view has a CONSEQUENT_VALUE column.
- Rule view when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as categorical, the view has a CONSEQUENT_VALUE column.

For the example that produces the following rules, see “Example: Calculating Aggregates” in *Oracle Machine Learning for SQL Concepts*.

The view reports two sets of aggregates results:

1. ANT_RULE_PROFIT refers to the total profit for the antecedent itemset with respect to the rule, the profit for each individual item of the antecedent itemset is shown in the ANTECEDENT(XMLtype) column, CON_RULE_PROFIT refers to the total profit for the consequent item with respect to the rule.

   In the example, for rule (A, B) => C, the rule itemset (A, B, C) occurs in the transactions of customer 1 and customer 3. The ANT_RULE_PROFIT is $21.20, The ANTECEDENT is shown as

follow, which tells that item A has profit 5.00 + 3.00 = $8.00 and item B has profit 3.20 + 10.00 = $13.20, which sum up to ANT_RULE_PROFIT.

```
<itemset NUMAGGR="1" ASSO_AGG0="profit"><item><item_name>A</
item_name><ASSO_AGG0>8.0E+000</ASSO_AGG0></item><item><item_name>B</
item_name><ASSO_AGG0>1.32E+001</ASSO_AGG0></item></itemset>
The CON_RULE_PROFIT is 12.00 + 14.00 = $26.00
```

2. ANT_PROFIT refers to the total profit for the antecedent itemset, while CON_PROFIT refers to the total profit for the consequent item. The difference between CON_PROFIT and CON_RULE_PROFIT (the same applies to ANT_PROFIT and ANT_RULE_PROFIT) is that CON_PROFIT counts all profit for the consequent item across all transactions where the consequent occurs, while CON_RULE_PROFIT only counts across transactions where the rule itemset occurs.

   For example, item C occurs in transactions for customer 1, 2 and 3, CON_PROFIT is 12.00 + 4.20 + 14.00 = $30.20, while CON_RULE_PROFIT only counts transactions for customer 1 and 3 where the rule itemset (A, B, C) occurs.

   Similarly, ANT_PROFIT counts all transactions where itemset (A, B) occurs, while ANT_RULE_PROFIT counts only transactions where the rule itemset (A, B, C) occurs. In this example, by coincidence, both count transactions for customer 1 and 3, and have the same value.

Example 4-16 Examples

The following example shows the view when setting ASSO_AGGREGATES specifies column profit and column sales to be aggregated. In this example, ITEM_VALUE column is not specified.

```
Name                                      Type
 ----------------------------------------- ----------------------------
 PARTITION_NAME                            VARCHAR2(128)
 RULE_ID                                   NUMBER
 RULE_SUPPORT                              NUMBER
 RULE_CONFIDENCE                           NUMBER
 RULE_LIFT                                 NUMBER
 RULE_REVCONFIDENCE                        NUMBER
 ANTECEDENT_SUPPORT                        NUMBER
 NUMBER_OF_ITEMS                           NUMBER
 CONSEQUENT_SUPPORT                        NUMBER
 CONSEQUENT_NAME                           VARCHAR2(4000)
 ANTECEDENT                                SYS.XMLTYPE
 ANT_RULE_PROFIT                           BINARY_DOUBLE
 CON_RULE_PROFIT                           BINARY_DOUBLE
 ANT_PROFIT                                BINARY_DOUBLE
 CON_PROFIT                                BINARY_DOUBLE
 ANT_RULE_SALES                            BINARY_DOUBLE
 CON_RULE_SALES                            BINARY_DOUBLE
 ANT_SALES                                 BINARY_DOUBLE
 CON_SALES                                 BINARY_DOUBLE
```

The rule view has a CONSEQUENT_VALUE column when ODMS_ITEM_ID_COLUMN_NAME is set and Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as numerical or categorical.

2-Dimensional Inputs

In Oracle Machine Learning for SQL, association models can be built using either transactional or two-dimensional data formats. For two-dimensional input, each item is defined by three fields: `NAME`, `VALUE` and `SUBNAME`. The `NAME` field is the name of the column. The `VALUE` field is the content of the column. The `SUBNAME` field is used when the input data table contains a nested table. In that case, `SUBNAME` is the name of the nested table's column. See, Example: Creating a Nested Column for Market Basket Analysis. In this example, there is a nested column. The `CONSEQUENT_SUBNAME` is the `ATTRIBUTE_NAME` part of the nested column. That is, `'O/S Documentation Set - English'` and `CONSEQUENT_VALUE` is the value part of the nested column, which is, 1.

The view uses three columns for the consequent. The rule view has the following columns:

```
Name                                      Type
 ---------------------- --------------------
 PARTITION_NAME                            VARCHAR2(128)
 RULE_ID                                   NUMBER
 RULE_SUPPORT                              NUMBER
 RULE_CONFIDENCE                           NUMBER
 RULE_LIFT                                 NUMBER
 RULE_REVCONFIDENCE                        NUMBER
 ANTECEDENT_SUPPORT                        NUMBER
 NUMBER_OF_ITEMS                           NUMBER
 CONSEQUENT_SUPPORT                        NUMBER
 CONSEQUENT_NAME                           VARCHAR2(4000)
 CONSEQUENT_SUBNAME                        VARCHAR2(4000)
 CONSEQUENT_VALUE                          VARCHAR2(4000)
 ANTECEDENT                                SYS.XMLTYPE
```

> **Note**
>
> All of the types for three columns for the consequent are `VARCHAR2`. `ASSO_AGGREGATES` is not applicable for 2-Dimensional input format.

The following table displays rule view columns for 2-Dimensional input with the descriptions of only the fields that are specific to 2-D inputs.

Table 4-17 Rule View for 2-Dimensional Input

Column Name	Description
`CONSEQUENT_SUBNAME`	For two-dimensional inputs, `CONSEQUENT_SUBNAME` is used for nested column in the input data table.
`CONSEQUENT_VALUE`	The value of the consequent when setting `Item_value` is set with `TYPE` as numerical or categorical.

Table 4-17 (Cont.) Rule View for 2-Dimensional Input

Column Name	Description
ANTECEDENT	The antecedent is described as an itemset. The itemset contains >= 1 items. Each item is defined using ITEM_NAME, ITEM_SUBNAME, and ITEM_VALUE: As an example, assuming that this is not a nested table input, and the antecedent contains one item: (name ADDR, value MA). The antecedent (XMLtype) is as follows: <itemset NUMAGGR="0"><item><item_name>ADDR</item_name><item_subname></item_subname><item_value>MA</item_value></item></itemset> For 2-Dimensional input with nested table, the subname field is filled.

Global Name-Value Pairs View for Association Rules

Global Name-Value Pairs View produces a single column for an association model. The following table describes the columns returned for association model.

Table 4-18 Global Name-Value Pairs View for an Association Model

Name	Description
ITEMSET_COUNT	The number of itemsets generated.
MAX_SUPPORT	The maximum support.
NUM_ROWS	The total number of rows used in the build.
RULE_COUNT	The number of association rules in the model generated.
TRANSACTION_COUNT	The number of the transactions in the input data.

4.9.2 Model Detail View for Frequent Itemsets

The model detail view DM$VI*model_name* contains information about frequent itemsets.

The Association Rule Itemsets view (DM$VI*model_name*) has the following columns:

```
Name              Type
-------------     ------------------
PARTITION_NAME    VARCHAR2 (128)
ITEMSET_ID        NUMBER
SUPPORT           NUMBER
NUMBER_OF_ITEMS   NUMBER
 ITEMSET           SYS.XMLTYPE
```

Table 4-19 Association Rule Itemsets View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model

Table 4-19 (Cont.) Association Rule Itemsets View

Column Name	Description
ITEMSET_ID	Itemset identifier
SUPPORT	Support of the itemset
NUMBER_OF_ITEMS	Number of items in the itemset
ITEMSET	Frequent itemset The structure of the SYS.XMLTYPE column itemset is the same as the corresponding Antecedent column of the rule view.

4.9.3 Model Detail Views for Transactional Itemsets

The model detail view DM$VT*model_name* contains information about the transactional itemsets.

For the very common case of transactional data without aggregates, the Association Rule Itemsets For Transactional Data view (DM$VT*model_name*) provides the itemsets information in transactional format. This view can help improve performance for some queries as compared to the view with the XML column. The transactional itemsets view has the following columns:

```
Name                 Type
-----------------    -----------------
PARTITION_NAME       VARCHAR2(128)
ITEMSET_ID           NUMBER
ITEM_ID              NUMBER
SUPPORT              NUMBER
NUMBER_OF_ITEMS      NUMBER
ITEM_NAME            VARCHAR2(4000)
```

Table 4-20 Association Rule Itemsets For Transactional Data View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
ITEMSET_ID	Itemset identifier
ITEM_ID	Item identifier
SUPPORT	Support of the itemset
NUMBER_OF_ITEMS	Number of items in the itemset
ITEM_NAME	The name of the item

4.9.4 Model Detail View for Transactional Rule

The model detail view DM$VA*model_name* contains information about transactional rules and transactional itemsets.

Transactional data without aggregates also has an Association Rules For Transactional Data view (DM$VA*model_name*). This view can improve performance for some queries as compared to the view with the XML column. The transactional rule view has the following columns:

```
Name                                    Type
--------------------------------------- ----------------------------
PARTITION_NAME                          VARCHAR2(128)
RULE_ID                                 NUMBER
ANTECEDENT_PREDICATE                    VARCHAR2(4000)
CONSEQUENT_PREDICATE                    VARCHAR2(4000)
RULE_SUPPORT                            NUMBER
RULE_CONFIDENCE                         NUMBER
RULE_LIFT                               NUMBER
RULE_REVCONFIDENCE                      NUMBER
RULE_ITEMSET_ID                         NUMBER
ANTECEDENT_SUPPORT                      NUMBER
CONSEQUENT_SUPPORT                      NUMBER
NUMBER_OF_ITEMS                         NUMBER
```

Table 4-21 Association Rules For Transactional Data View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
RULE_ID	Rule identifier
ANTECEDENT_PREDICATE	Name of the Antecedent item.
CONSEQUENT_PREDICATE	Name of the Consequent item
RULE_SUPPORT	Support of the rule
RULE_CONFIDENCE	The likelihood a transaction satisfies the rule when it contains the Antecedent.
RULE_LIFT	The degree of improvement in the prediction over random chance when the rule is satisfied
RULE_REVCONFIDENCE	The number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs
RULE_ITEMSET_ID	Itemset identifier
ANTECEDENT_SUPPORT	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions
CONSEQUENT_SUPPORT	The ratio of the number of transactions that satisfy the consequent to the total number of transactions
NUMBER_OF_ITEMS	Number of items in the rule

4.9.5 Model Detail Views for Classification Algorithms

Model detail views for classification algorithms are the target map view and scoring cost view, which are applicable to all classification algorithms.

These are the available model views for Classification algorithm:

Model Views	Description
DM$VA*model_name*	Variable Importance
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*:	Model Build Alerts

The Classification Targets view (DM$VT*model_name*) describes the target distribution for classification models. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_VALUE                              NUMBER/VARCHAR2
TARGET_COUNT                              NUMBER
TARGET_WEIGHT                             NUMBER
```

Table 4-22 Classification Targets View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
TARGET_VALUE	Target value, numerical or categorical
TARGET_COUNT	Number of rows for a given TARGET_VALUE
TARGET_WEIGHT	Weight for a given TARGET_VALUE

The Scoring Cost Matrix view (DM$VC*model_name*) describes the scoring cost matrix for classification models. The view has the following columns:

```
Name                                      Type
----------------------------------------- ------------------------------
PARTITION_NAME                            VARCHAR2(128)
ACTUAL_TARGET_VALUE                       NUMBER/VARCHAR2
PREDICTED_TARGET_VALUE                    NUMBER/VARCHAR2
COST                                      NUMBER
```

Table 4-23 Scoring Cost Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model

Table 4-23 (Cont.) Scoring Cost Matrix View

Column Name	Description
ACTUAL_TARGET_VALUE	A valid target value
PREDICTED_TARGET_VALUE	Predicted target value
COST	Associated cost for the actual and predicted target value pair

4.9.6 Model Detail Views for CUR Matrix Decomposition

Model detail views for CUR Matrix Decomposition contain information about the scores and ranks of attributes and rows.

CUR Matrix Decomposition models have the following views:

Attribute importance and rank: DM$VC*model_name*

Row importance and rank: DM$VR*model_name*

Global statistics: DM$VG

The attribute importance and rank view DM$VC*model_name* has the following columns:

```
Name                           Type
-----------------              -----------------
PARTITION_NAME                 VARCHAR2(128)
ATTRIBUTE_NAME                 VARCHAR2(128)
ATTRIBUTE_SUBNAME              VARCHAR2(4000)
ATTRIBUTE_VALUE                VARCHAR2(4000)
ATTRIBUTE_IMPORTANCE           NUMBER
ATTRIBUTE_RANK                 NUMBER
```

Table 4-24 Attribute Importance and Rank View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Attribute name
ATTRIBUTE_SUBNAME	Attribute subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Value of the attribute
ATTRIBUTE_IMPORTANCE	Attribute leverage score
ATTRIBUTE_RANK	Attribute rank based on leverage score

The view DM$VR*model_name* exposes the leverage scores and ranks of all selected rows through a view. This view is created when users decide to perform row importance and the CASE_ID column is present. The view has the following columns:

```
Name                    Type
---------------------     -----------------------
PARTITION_NAME            VARCHAR2(128)
CASE_ID                   Original cid data types,
                          including NUMBER, VARCHAR2,
```

```
                                DATE, TIMESTAMP,
                                TIMESTAMP WITH TIME ZONE,
                                TIMESTAMP WITH LOCAL TIME ZONE
ROW_IMPORTANCE                  NUMBER
ROW_RANK                        NUMBER
```

Table 4-25 Row Importance and Rank View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Case ID. The supported case ID types are the same as that supported for GLM, SVD, and ESA algorithms.
ROW_IMPORTANCE	Row leverage score
ROW_RANK	Row rank based on leverage score

The following table describes global statistics for CUR Matrix Decomposition.

Table 4-26 CUR Matrix Decomposition Statistics Information In Model Global View.

Name	Description
NUM_COMPONENTS	Number of SVD components (SVD rank)
NUM_ROWS	Number of rows used in the model build

4.9.7 Model Detail Views for Decision Tree

The model detail views specific to Decision Tree are the hierarchy view, node statistics view, node description view, and the cost matrix view.

These are the model views available for Decision Tree:

Model Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	Decision Tree Statistics
DM$VM*model_name*	Decision Tree Build Cost Matrix
DM$VO*model_name*	Decision Tree Nodes
DM$VP*model_name*	Decision Tree Hierarchy
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The Decision Tree Hierarchy view (DM$VP*model_name*) describes the decision tree hierarchy and the split information for each level in the decision tree. The view has the following columns:

```
Name                            Type
------------------------------- ---------------------------
PARTITION_NAME                  VARCHAR2(128)
PARENT                          NUMBER
```

```
SPLIT_TYPE                     VARCHAR2
NODE                           NUMBER
ATTRIBUTE_NAME                 VARCHAR2(128)
ATTRIBUTE_SUBNAME              VARCHAR2(4000)
OPERATOR                       VARCHAR2
VALUE                          SYS.XMLTYPE
```

Table 4-27 Decision Tree Hierarchy View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
PARENT	Node ID of the parent
SPLIT_TYPE	The main or surrogate split
NODE	The node ID
ATTRIBUTE_NAME	The attribute used as the splitting criterion at the parent node to produce this node.
ATTRIBUTE_SUBNAME	Split attribute subname. The value is null for non-nested columns.
OPERATOR	Split operator
VALUE	Value used as the splitting criterion. This is an XML element described using the <Element> tag. For example, <Element>Windy</Element><Element>Hot</Element>.

The Decision Tree Statistics view (DM$VI*model_name*) describes the statistics associated with individual tree nodes. The statistics include a target histogram for the data in the node. The view has the following columns:

```
Name                           Type
------------------------------ ----------------------------
PARTITION_NAME                 VARCHAR2(128)
NODE                           NUMBER
NODE_SUPPORT                   NUMBER
PREDICTED_TARGET_VALUE         NUMBER/VARCHAR2
TARGET_VALUE                   NUMBER/VARCHAR2
TARGET_SUPPORT                 NUMBER
```

Table 4-28 Decision Tree Statistics View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
NODE	The node ID
NODE_SUPPORT	Number of records in the training set that belong to the node
PREDICTED_TARGET_VALUE	Predicted Target value
TARGET_VALUE	A target value seen in the training data
TARGET_SUPPORT	The number of records that belong to the node and have the value specified in the TARGET_VALUE column

The Decision Tree Nodes (DM$V0*model_name*) view describes higher level node. The DM$V0*model_name* has the following columns:

```
Name                              Type
--------------------------------- ---------------------------
PARTITION_NAME                    VARCHAR2(128)
NODE                              NUMBER
NODE_SUPPORT                      NUMBER
PREDICTED_TARGET_VALUE            NUMBER/VARCHAR2
PARENT                            NUMBER
ATTRIBUTE_NAME                    VARCHAR2(128)
ATTRIBUTE_SUBNAME                 VARCHAR2(4000)
OPERATOR                          VARCHAR2
VALUE                             SYS.XMLTYPE
```

Table 4-29 Decision Tree Nodes View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
NODE	The node ID
NODE_SUPPORT	Number of records in the training set that belong to the node
PREDICTED_TARGET_VALUE	Predicted Target value
PARENT	The ID of the parent
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
OPERATOR	Attribute predicate operator - a conditional operator taking the following values: *IN*, = , <>, < , >, <=, and >=
VALUE	Value used as the description criterion. This is an XML element described using the `<Element>` tag. For example, `<Element>Windy</Element><Element>Hot</Element>`.

The Decision Tree Build Cost Matrix view (DM$VM*model_name*) describes the cost matrix used by the Decision Tree build. The DM$VM*model_name* view has the following columns:

```
Name                                    Type
--------------------------------------- -------------------------------
PARTITION_NAME                          VARCHAR2(128)
ACTUAL_TARGET_VALUE                     NUMBER/VARCHAR2
PREDICTED_TARGET_VALUE                  NUMBER/VARCHAR2
COST                                    NUMBER
```

Table 4-30 Decision Tree Build Cost Matrix View

Parameter	Description
PARTITION_NAME	Partition name in a partitioned model
ACTUAL_TARGET_VALUE	Valid target value

Table 4-30 (Cont.) Decision Tree Build Cost Matrix View

Parameter	Description
PREDICTED_TARGET_VALUE	Predicted Target value
COST	Associated cost for the actual and predicted target value pair

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) columns specific to a Decision Tree model.

Table 4-31 Global Name-Value Pairs View

Name	Description
NUM_ROWS	The total number of rows used in the build

4.9.8 Model Detail Views for Generalized Linear Model

Model detail views specific to Generalized Linear Model (GLM) such as details and row diagnostics for linear and logistic regression models are discussed.

The following model views are available for GLM:

Model Views	Description
DM$VA*model_name*	GLM Regression Row Diagnostics
DM$VD*model_name*	GLM Regression Attribute Diagnostics
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The GLM Regression Attribute Diagnostics view (DM$VD*model_name*) describes the final model information for both linear regression models and logistic regression models.

For linear regression, the view DM$VD*model_name* has the following columns:

```
Name                              Type
--------------------------------- ---------------------------
PARTITION_NAME                    VARCHAR2(128)
ATTRIBUTE_NAME                    VARCHAR2(128)
ATTRIBUTE_SUBNAME                 VARCHAR2(4000)
ATTRIBUTE_VALUE                   VARCHAR2(4000)
FEATURE_EXPRESSION                VARCHAR2(4000)
COEFFICIENT                       BINARY_DOUBLE
STD_ERROR                         BINARY_DOUBLE
TEST_STATISTIC                    BINARY_DOUBLE
P_VALUE                           BINARY_DOUBLE
VIF                               BINARY_DOUBLE
STD_COEFFICIENT                   BINARY_DOUBLE
LOWER_COEFF_LIMIT                 BINARY_DOUBLE
UPPER_COEFF_LIMIT                 BINARY_DOUBLE
```

For logistic regression, the view DM$VD*model_name* has the following columns:

```
Name                                Type
----------------------------------- --------------------------
PARTITION_NAME                      VARCHAR2(128)
TARGET_VALUE                        NUMBER/VARCHAR2
ATTRIBUTE_NAME                      VARCHAR2(128)
ATTRIBUTE_SUBNAME                   VARCHAR2(4000)
ATTRIBUTE_VALUE                     VARCHAR2(4000)
FEATURE_EXPRESSION                  VARCHAR2(4000)
COEFFICIENT                         BINARY_DOUBLE
STD_ERROR                           BINARY_DOUBLE
TEST_STATISTIC                      BINARY_DOUBLE
P_VALUE                             BINARY_DOUBLE
STD_COEFFICIENT                     BINARY_DOUBLE
LOWER_COEFF_LIMIT                   BINARY_DOUBLE
UPPER_COEFF_LIMIT                   BINARY_DOUBLE
EXP_COEFFICIENT                     BINARY_DOUBLE
EXP_LOWER_COEFF_LIMIT               BINARY_DOUBLE
EXP_UPPER_COEFF_LIMIT               BINARY_DOUBLE
```

Table 4-32 Model View for Linear and Logistic Regression Models

Column Name	Description
PARTITION_NAME	The name of a feature in the model
TARGET_VALUE	Valid target value
ATTRIBUTE_NAME	The attribute name when there is no subname, or first part of the attribute name when there is a subname. ATTRIBUTE_NAME is the name of a column in the source table or view. If the column is a non-nested, numeric column, then ATTRIBUTE_NAME is the name of the machine learning attribute. For the intercept, ATTRIBUTE_NAME is null. Intercepts are equivalent to the bias term in SVM models.
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns. When the nested column is numeric, the machine learning attribute is identified by the combination ATTRIBUTE_NAME - ATTRIBUTE_SUBNAME. If the column is not nested, ATTRIBUTE_SUBNAME is null. If the attribute is an intercept, both the ATTRIBUTE_NAME and the ATTRIBUTE_SUBNAME are null.
ATTRIBUTE_VALUE	A unique value that can be assumed by a categorical column or nested categorical column. For categorical columns, a machine learning attribute is identified by a unique ATTRIBUTE_NAME.ATTRIBUTE_VALUE pair. For nested categorical columns, a machine learning attribute is identified by the combination: ATTRIBUTE_NAME.ATTRIBUTE_SUBNAME.ATTRIBUTE_VALUE. For numerical attributes, ATTRIBUTE_VALUE is null.

Table 4-32 (Cont.) Model View for Linear and Logistic Regression Models

Column Name	Description
FEATURE_EXPRESSION	The feature name constructed by the algorithm when feature selection is enabled. If feature selection is not enabled, the feature name is the fully-qualified attribute name (*attribute_name.attribute_subname* if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form: *fully-qualified_attribute_name.attribute_value* When feature generation is enabled, a term in the model can be a single machine learning attribute or the product of up to 3 machine learning attributes. Component machine learning attributes can be repeated within a single term. If feature generation is not enabled or, if feature generation is enabled, but no multiple component terms are discovered by the CREATE model process, then FEATURE_EXPRESSION is null. **Note** In 12*c* Release 2, the algorithm does not subtract the mean from numerical components.
COEFFICIENT	The estimated coefficient.
STD_ERROR	Standard error of the coefficient estimate.
TEST_STATISTIC	For linear regression, the t-value of the coefficient estimate. For logistic regression, the Wald chi-square value of the coefficient estimate.
P_VALUE	Probability of the TEST_STATISTIC under the (NULL) hypothesis that the term in the model is not statistically significant. A low probability indicates that the term is significant, while a high probability indicates that the term can be better discarded. Used to analyze the significance of specific attributes in the model.
VIF	Variance Inflation Factor. The value is zero for the intercept. For logistic regression, VIF is null.
STD_COEFFICIENT	Standardized estimate of the coefficient.
LOWER_COEFF_LIMIT	Lower confidence bound of the coefficient.
UPPER_COEFF_LIMIT	Upper confidence bound of the coefficient.
EXP_COEFFICIENT	Exponentiated coefficient for logistic regression. For linear regression, EXP_COEFFICIENT is null.
EXP_LOWER_COEFF_LIMIT	Exponentiated coefficient for lower confidence bound of the coefficient for logistic regression. For linear regression, EXP_LOWER_COEFF_LIMIT is null.
EXP_UPPER_COEFF_LIMIT	Exponentiated coefficient for upper confidence bound of the coefficient for logistic regression. For linear regression, EXP_UPPER_COEFF_LIMIT is null.

The GLM Regression Row Diagnostics view DM$VA*model_name* describes row level information for both linear regression models and logistic regression models. For linear regression, the view DM$VA*model_name* has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
PARTITION_NAME                    VARCHAR2(128)
CASE_ID                           NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                  TIMESTAMP WITH TIME ZONE,
                                  TIMESTAMP WITH LOCAL TIME ZONE
TARGET_VALUE                      BINARY_DOUBLE
PREDICTED_TARGET_VALUE            BINARY_DOUBLE
Hat                               BINARY_DOUBLE
RESIDUAL                          BINARY_DOUBLE
STD_ERR_RESIDUAL                  BINARY_DOUBLE
STUDENTIZED_RESIDUAL              BINARY_DOUBLE
PRED_RES                          BINARY_DOUBLE
COOKS_D                           BINARY_DOUBLE
```

Table 4-33 GLM Regression Row Diagnostics View for Linear Regression

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Name of the case identifier
TARGET_VALUE	The actual target value as taken from the input row
PREDICTED_TARGET_VALUE	The model predicted target value for the row
HAT	The diagonal element of the n*n (n=number of rows) that the Hat matrix identifies with a specific input row. The model predictions for the input data are the product of the Hat matrix and vector of input target values. The diagonal elements (Hat values) represent the influence of the i^{th} row on the i^{th} fitted value. Large Hat values are indicators that the i^{th} row is a point of high leverage, a potential outlier.
RESIDUAL	The difference between the predicted and actual target value for a specific input row.
STD_ERR_RESIDUAL	The standard error residual, sometimes called the Studentized residual, re-scales the residual to have constant variance across all input rows in an effort to make the input row residuals comparable. The process multiplies the residual by square root of the row weight divided by the product of the model mean square error and 1 minus the Hat value.
STUDENTIZED_RESIDUAL	Studentized deletion residual adjusts the standard error residual for the influence of the current row.
PRED_RES	The predictive residual is the weighted square of the deletion residuals, computed as the row weight multiplied by the square of the residual divided by 1 minus the Hat value.
COOKS_D	Cook's distance is a measure of the combined impact of the i^{th} case on all of the estimated regression coefficients.

For logistic regression, the view DM$VA*model_name* has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
PARTITION_NAME                    VARCHAR2(128)
```

CASE_ID	NUMBER/VARHCAR2, DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE
TARGET_VALUE	NUMBER/VARCHAR2
TARGET_VALUE_PROB	BINARY_DOUBLE
Hat	BINARY_DOUBLE
WORKING_RESIDUAL	BINARY_DOUBLE
PEARSON_RESIDUAL	BINARY_DOUBLE
DEVIANCE_RESIDUAL	BINARY_DOUBLE
C	BINARY_DOUBLE
CBAR	BINARY_DOUBLE
DIFDEV	BINARY_DOUBLE
DIFCHISQ	BINARY_DOUBLE

Table 4-34 GLM Regression Row Diagnostics View for Logistic Regression

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Name of the case identifier
TARGET_VALUE	The actual target value as taken from the input row
TARGET_VALUE_PROB	Model estimate of the probability of the predicted target value.
Hat	The Hat value concept from linear regression is extended to logistic regression by multiplying the linear regression Hat value by the variance function for logistic regression, the predicted probability multiplied by 1 minus the predicted probability.
WORKING_RESIDUAL	The working residual is the residual of the working response. The working response is the response on the linearized scale. For logistic regression it has the form: the i^{th} row residual divided by the variance of the i^{th} row prediction. The variance of the prediction is the predicted probability multiplied by 1 minus the predicted probability. WORKING_RESIDUAL is the difference between the working response and the linear predictor at convergence.
PEARSON_RESIDUAL	The Pearson residual is a re-scaled version of the working residual, accounting for the weight. For logistic regression, the Pearson residual multiplies the residual by a factor that is computed as square root of the weight divided by the variance of the predicted probability for the i^{th} row. RESIDUAL is 1 minus the predicted probability of the actual target value for the row.
DEVIANCE_RESIDUAL	The DEVIANCE_RESIDUAL is the contribution to the model deviance of the i^{th} observation. For logistic regression it has the form the square root of 2 times the log(1 + e^eta) - eta for the non-reference class and - square root of 2 time the log (1 + eta) for the reference class, where eta is the linear prediction (the prediction as if the model were a linear regression).
C	Measures the overall change in the fitted logits due to the deletion of the i^{th} observation for all points including the one deleted (the i^{th} point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by the square of 1 minus the Hat value. Confidence interval displacement diagnostics that provides scalar measure of the influence of individual observations.

Table 4-34 (Cont.) GLM Regression Row Diagnostics View for Logistic Regression

Column Name	Description
CBAR	C and CBAR are extensions of Cooks' distance for logistic regression. CBAR measures the overall change in the fitted logits due to the deletion of the i^{th} observation for all points excluding the one deleted (the i^{th} point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by (1 minus the Hat value) Confidence interval displacement diagnostic which measures the influence of deleting an individual observation.
DIFDEV	A statistic that measures the change in deviance that occurs when an observation is deleted from the input. It is computed as the square of the deviance residual plus CBAR.
DIFCHISQ	A statistic that measures the change in the Pearson chi-square statistic that occurs when an observation is deleted from the input. It is computed as CBAR divided by the Hat value.

Global Details for GLM: Linear Regression

The following table describes Global Name-Value Pairs (DM$VG) for a linear regression model.

Table 4-35 Global Details for Linear Regression

Name	Description
ADJUSTED_R_SQUARE	Adjusted R-Square
AIC	Akaike's information criterion
COEFF_VAR	Coefficient of variation
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
CORRECTED_TOTAL_DF	Corrected total degrees of freedom
CORRECTED_TOT_SS	Corrected total sum of squares
DEPENDENT_MEAN	Dependent mean
ERROR_DF	Error degrees of freedom
ERROR_MEAN_SQUARE	Error mean square
ERROR_SUM_SQUARES	Error sum of squares
F_VALUE	Model *F* value statistic
GMSEP	Estimated mean square error of the prediction, assuming multivariate normality
HOCKING_SP	Hocking Sp statistic
ITERATIONS	Tracks the number of SGD iterations. Applicable only when the solver is SGD.
J_P	JP statistic (the final prediction error)
MODEL_DF	Model degrees of freedom
MODEL_F_P_VALUE	Model *F* value probability

Table 4-35 (Cont.) Global Details for Linear Regression

Name	Description
MODEL_MEAN_SQUARE	Model mean square error
MODEL_SUM_SQUARES	Model sum of square errors
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)
NUM_ROWS	Number of rows
R_SQ	R-Square
RANK_DEFICIENCY	The number of predictors excluded from the model due to multi-collinearity
ROOT_MEAN_SQ	Root mean square error
SBIC	Schwarz's Bayesian information criterion

Global Details for GLM: Logistic Regression

The following table returns Global Name-Value Pairs (DM$VG) for a logistic regression model.

Table 4-36 Global Details for Logistic Regression

Name	Description
AIC_INTERCEPT	Akaike's criterion for the fit of the baseline, intercept-only, model
AIC_MODEL	Akaike's criterion for the fit of the intercept and the covariates (predictors) mode
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
DEPENDENT_MEAN	Dependent mean
ITERATIONS	Tracks the number of SGD iterations (number of IRLS iterations). Applicable only when the solver is SGD.
LR_DF	Likelihood ratio degrees of freedom
LR_CHI_SQ	Likelihood ratio chi-square value
LR_CHI_SQ_P_VALUE	Likelihood ratio chi-square probability value
NEG2_LL_INTERCEPT	-2 log likelihood of the baseline, intercept-only, model
NEG2_LL_MODEL	-2 log likelihood of the model
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)
NUM_ROWS	Number of rows
PCT_CORRECT	Percent of correct predictions
PCT_INCORRECT	Percent of incorrectly predicted rows
PCT_TIED	Percent of cases where the estimated probabilities are equal for both target classes
PSEUDO_R_SQ_CS	Pseudo R-square Cox and Snell
PSEUDO_R_SQ_N	Pseudo R-square Nagelkerke

Table 4-36 (Cont.) Global Details for Logistic Regression

Name	Description
RANK_DEFICIENCY	The number of predictors excluded from the model due to multi-collinearity
SC_INTERCEPT	Schwarz's Criterion for the fit of the baseline, intercept-only, model
SC_MODEL	Schwarz's Criterion for the fit of the intercept and the covariates (predictors) model

Note

- When ridge regression is enabled, fewer global details are returned. For information about ridge, see *Oracle Machine Learning for SQL Concepts*.
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*
- Model Detail Views for Global Information
  Model detail views for global information contain information about global statistics, alerts, and computed settings.

4.9.9 Model Detail View for Multivariate State Estimation Technique - Sequential Probability Ratio Test

The model detail view specific to Multivariate State Estimation Technique - Sequential Probability Ratio Test contains information about Global Name-Value Paris.

The following are the available model views for MSET-SPRT:

Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The following table lists the Global Name-Value Pairs (DM$VG*model_name*) for an MSET-SPRT. This statistic is included when due to memory constraints MSET-SPRT cannot use the MSET_MEMORY_VECTORS value set by the user.

Table 4-37 MSET-SPRT Information in the Model Global View

Name	Description
NUM_MVEC	The number of memory vectors used by the model.

4.9.10 Model Detail Views for Naive Bayes

The model detail views specific to Naive Bayes are the prior view and result view.

These the model views available for Naive Bayes:

Model Views	Description
DM$VB*model_name*	Automatic Data Preparation Binning
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VP*model_name*	Naive Bayes Target Priors
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VV*model_name*	Naive Bayes Conditional Probabilities
DM$VW*model_name*	Model Build Alerts

The Naive Bayes Target Priors view (DM$VP*model_name*) describes the priors of the targets for a Naive Bayes model. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_NAME                               VARCHAR2(128)
TARGET_VALUE                              NUMBER/VARCHAR2
PRIOR_PROBABILITY                         BINARY_DOUBLE
COUNT                                     NUMBER
```

Table 4-38 Naive Bayes Target Priors View for Naive Bayes

Column Name	Description
PARTITION_NAME	The name of a feature in the model
TARGET_NAME	Name of the target column
TARGET_VALUE	Target value, numerical or categorical
PRIOR_PROBABILITY	Prior probability for a given TARGET_VALUE
COUNT	Number of rows for a given TARGET_VALUE

The Naive Bayes Conditional Probabilities view (DM$VV*model_view*) describes the conditional probabilities of the Naive Bayes model. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
TARGET_NAME                               VARCHAR2(128)
```

TARGET_VALUE	NUMBER/VARCHAR2
ATTRIBUTE_NAME	VARCHAR2(128)
ATTRIBUTE_SUBNAME	VARCHAR2(4000)
ATTRIBUTE_VALUE	VARCHAR2(4000)
CONDITIONAL_PROBABILITY	BINARY_DOUBLE
COUNT	NUMBER

Table 4-39 Naive Bayes Conditional Probabilities View for Naive Bayes

Column Name	Description
PARTITION_NAME	The name of a feature in the model
TARGET_NAME	Name of the target column
TARGET_VALUE	Target value, numerical or categorical
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Machine learning attribute value for the column ATTRIBUTE_NAME or the nested column ATTRIBUTE_SUBNAME (if any).
CONDITIONAL_PROBABILITY	Conditional probability of a machine learning attribute for a given target
COUNT	Number of rows for a given machine learning attribute and a given target

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to a Naive Bayes model.

Table 4-40 Global Name-Value Pairs View for Naive Bayes

Name	Description
NUM_ROWS	The total number of rows used in the build

4.9.11 Model Detail Views for Neural Network

Model detail views specific to Neural Network contain information about the weights of the neurons: input layer and hidden layers.

These are the model views available for Neural Network:

Model Views	Description
DM$VA*model_name*	Neural Network Weights
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The Neural Network Weights view (DM$VA*model_name*) has the following columns:

```
Name
Type
----------------------          ----------------------
PARTITION_NAME                  VARCHAR2(128)
LAYER                           NUMBER
IDX_FROM                        NUMBER
ATTRIBUTE_NAME                  VARCHAR2(128)
ATTRIBUTE_SUBNAME               VARCHAR2(4000)
ATTRIBUTE_VALUE                 VARCHAR2(4000)
IDX_TO                          NUMBER
TARGET_VALUE                    NUMBER/VARCHAR2
WEIGHT                          BINARY_DOUBLE
```

Table 4-41 Neural Network Weights View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
LAYER	Layer ID, 0 as an input layer
IDX_FROM	Node index that the weight connects from (attribute id for input layer)
ATTRIBUTE_NAME	Attribute name (only for the input layer)
ATTRIBUTE_SUBNAME	Attribute subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
IDX_TO	Node index that the weights connects to
TARGET_VALUE	Target value. The value is null for regression.
WEIGHT	Value of the weight

The view Global Name-Value Pairs (DM$VG*model_name*) is a pre-existing view. The following name-value pairs are specific to a Neural Network view.

Table 4-42 Global Name-Value Pairs Viewfor Neural Network

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
ITERATIONS	Number of iterations
LOSS_VALUE	Loss function value (if it is with NNET_REGULARIZER_HELDASIDE regularization, it is the loss function value on test data)
NUM_ROWS	Number of rows in the model (or partitioned model)

4.9.12 Model Detail Views for Random Forest

Model detail views specific to Random Forest contain variable importance measures and statistics.

The following model detail views are available for Random Forest:

Model View	Description
DM$VA*model_name*	Variable Importance
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

Model detail views and statistics specific to Random Forest are:

- Variable Importance statistics DM$VA*model_name*
- Random Forest statistics in the Global Name-Value Pairs DM$VG*model_name* view

One of the important outputs from a Random Forest model build is a ranking of attributes based on their relative importance. This is measured using Mean Decrease Gini. The DM$VA*model_name* view has the following columns:

```
Name                          Type
----------------------        --------------
PARTITION_NAME                VARCHAR2(128)
ATTRIBUTE_NAME                VARCHAR2(128)
ATTRIBUTE_SUBNAME             VARCHAR2(128)
ATTRIBUTE_IMPORTANCE          BINARY_DOUBLE
```

Table 4-43 Variable Importance Model View

Column Name	Description
PARTITION_NAME	Partition name. The value is null for models which are not partitioned.
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_IMPORTANCE	Measure of importance for an attribute in the forest (mean Decrease Gini value)

The Global Name-Value Pairs (DM$VG*model_name*) view is a pre-existing view. The following name-value pairs are added to the view.

Table 4-44 Random Forest Statistics Information In Model Global View

Name	Description
AVG_DEPTH	Average depth of the trees in the forest
AVG_NODECOUNT	Average number of nodes per tree
MAX_DEPTH	Maximum depth of the trees in the forest
MAX_NODECOUNT	Maximum number of nodes per tree
MIN_DEPTH	Minimum depth of the trees in the forest
MIN_NODECOUNT	Minimum number of nodes per tree
NUM_ROWS	The total number of rows used in the build

4.9.13 Model Detail View for Support Vector Machine

Model detail views specific to Support Vector Machine (SVM) contain linear coefficients and support vector statistics.

These model views are available for SVM:

Model Views	Description
DM$VCS*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The linear coefficient view DM$VL*model_name* describes the coefficients of a linear SVM algorithm. The *target_value* field in the view is present only for classification and has the type of the target. Regression models do not have a *target_value* field.

The *reversed_coefficient* field shows the value of the coefficient after reversing the automatic data preparation transformations. If data preparation is disabled, then *coefficient* and *reversed_coefficient* have the same value. The view has the following columns:

```
Name                                    Type
--------------------------------------- --------------------------------
PARTITION_NAME                          VARCHAR2(128)
TARGET_VALUE                            NUMBER/VARCHAR2
ATTRIBUTE_NAME                          VARCHAR2(128)
ATTRIBUTE_SUBNAME                       VARCHAR2(4000)
ATTRIBUTE_VALUE                         VARCHAR2(4000)
COEFFICIENT                             BINARY_DOUBLE
REVERSED_COEFFICIENT                    BINARY_DOUBLE
```

Table 4-45 Linear Coefficient View for Support Vector Machine

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model

Table 4-45 (Cont.) Linear Coefficient View for Support Vector Machine

Column Name	Description
TARGET_VALUE	Target value, numerical or categorical
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Value of a categorical attribute
COEFFICIENT	Projection coefficient value
REVERSED_COEFFICIENT	Coefficient transformed on the original scale

The following table describes the SVM statistics global view.

Table 4-46 Support Vector Statistics Information In Model Global View

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance: • YES • NO
ITERATIONS	Number of iterations performed during build
NUM_ROWS	Number of rows used for the build
REMOVED_ROWS_ZERO_NORM	Number of rows removed due to 0 norm. This applies to one-class linear models only.

4.9.14 Model Detail Views for XGBoost

The model detail views specific to XGBoost contain information about Feature Importance view and Global Name-Value Pairs view.

The following are the available model views for XGBoost Classification:

Model Views	Description
DM$VC*model_name*	Scoring Cost Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	XGBoost Attribute Importance
DM$VS*model_name*	Computed Settings
DM$VT*model_name*	Classification Targets
DM$VW*model_name*	Model Build Alerts

The following are the available model views for XGBoost Regression:

Views	Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	XGBoost Attribute Importance
DM$VS*model_name*	Computed Settings

Views	Description
DM$VW*model_name*	Model Build Alerts

The DM$VI*model_name* view reports the feature importance values for each attribute of each partition of the model.

The view has the following columns for tree models (gbtree and dart boosters).

```
Name                Type
------------------  -------------
PNAME               VARCHAR2(128)
ATTRIBUTE_NAME      VARCHAR2(128)
ATTRIBUTE_SUBNAME   VARCHAR2(4000)
ATTRIBUTE_VALUE     VARCHAR2(4000)
GAIN                BINARY_DOUBLE
COVER               BINARY_DOUBLE
FREQUENCY           BINARY_DOUBLE
```

Table 4-47 Feature Importance View for a Tree Model

Column Name	Description
PNAME	The name of a partition in a partitioned model.
ATTRIBUTE_NAME	The column name.
ATTRIBUTE_SUBNAME	The nested column subname; the value is null for non-nested columns.
ATTRIBUTE_VALUE	The value of a categorical attribute.
GAIN	The fractional contribution of each feature to the model based on the total gain of a feature's splits; a higher percentage means a more important predictive feature.
COVER	The number of observation either seen by a split or collected by a leaf during training.
FREQUENCY	A percentage representing the relative number of times a feature has been used in trees.

For a linear model (gblinear) booster, the feature importance is the absolute magnitude of linear coefficients.

The view has the following columns for linear models.

```
Name          Type
------------------  -------------
PNAME               VARCHAR2(128)
ATTRIBUTE_NAME      VARCHAR2(128)
ATTRIBUTE_SUBNAME   VARCHAR2(4000)
ATTRIBUTE_VALUE     VARCHAR2(4000)
WEIGHT              BINARY_DOUBLE
CLASS               BINARY_DOUBLE
```

Table 4-48 Feature Importance View for a Linear Model

Column Name	Description
PNAME	The name of a partition in a partitioned model.
ATTRIBUTE_NAME	The column name.
ATTRIBUTE_SUBNAME	The nested column subname; the value is null for non-nested columns.
ATTRIBUTE_VALUE	The value of a categorical attribute.
WEIGHT	The linear coefficient of the feature.
CLASS	The class label for a multiclass model.

The DM$VG*model_name* view reports global statistics for an XGBoost model. The statistics include an evaluation of the training data set using the evaluation metric you specified with the learning task eval_metric setting, or the default eval_metric if you didn't specify one. The view displays only the result of the last training iteration. When you specify more than one eval_metric, the view contains multiple rows, one for each eval_metric.

4.9.15 Model Detail Views for Clustering Algorithms

Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

All clustering algorithms share the following views:

Model Views	Description
DM$VD*model_name*:	Clustering Description
DM$VA*model_name*	Clustering Attribute Statistics
DM$VH*model_name*	Clustering Histograms
DM$VR*model_name*	Clustering Rules

The Cluster Description view DM$VD*model_name* describes cluster level information about a clustering model. The view has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
PARTITION_NAME                    VARCHAR2(128)
CLUSTER_ID                        NUMBER
CLUSTER_NAME                      NUMBER/VARCHAR2
RECORD_COUNT                      NUMBER
PARENT                            NUMBER
TREE_LEVEL                        NUMBER
LEFT_CHILD_ID                     NUMBER
RIGHT_CHILD_ID                    NUMBER
```

Table 4-49 Clustering Description View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model

Table 4-49 (Cont.) Clustering Description View

Column Name	Description
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
RECORD_COUNT	Specifies the number of records
PARENT	The ID of the parent
TREE_LEVEL	Specifies the number of splits from the root
LEFT_CHILD_ID	The ID of the child cluster on the left side of the split
RIGHT_CHILD_ID	The ID of the child cluster on the right side of the split

The attribute view DM$VA*model_name* describes attribute level information about a clustering model. The values of the mean, variance, and mode for a particular cluster can be obtained from this view. The view has the following columns:

```
Name                                 Type
---------------------------------- ----------------------------
PARTITION_NAME                       VARCHAR2(128)
CLUSTER_ID                           NUMBER
CLUSTER_NAME                         NUMBER/VARCHAR2
ATTRIBUTE_NAME                       VARCHAR2(128)
ATTRIBUTE_SUBNAME                    VARCHAR2(4000)
MEAN                                 BINARY_DOUBLE
VARIANCE                             BINARY_DOUBLE
MODE_VALUE                           VARCHAR2(4000)
```

Table 4-50 Clustering Attribute Statistics

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname. For vector data types, this attribute shows each vector dimension as an individual predictor with DM$$xxx, where xxx is the position of the vector.
MEAN	The field returns the average value of a numeric attribute
VARIANCE	The variance of a numeric attribute
MODE_VALUE	The mode is the most frequent value of a categorical attribute

The histogram view DM$VH*model_name* describes histogram level information about a clustering model. The bin information as well as bin counts can be obtained from this view. The view has the following columns:

```
Name                                 Type
---------------------------------- ----------------------------
PARTITION_NAME                       VARCHAR2(128)
```

```
CLUSTER_ID                     NUMBER
CLUSTER_NAME                   NUMBER/VARCHAR2
ATTRIBUTE_NAME                 VARCHAR2(128)
ATTRIBUTE_SUBNAME              VARCHAR2(4000)
BIN_ID                         NUMBER
LOWER_BIN_BOUNDARY             BINARY_DOUBLE
UPPER_BIN_BOUNDARY             BINARY_DOUBLE
ATTRIBUTE_VALUE                VARCHAR2(4000)
COUNT                          NUMBER
```

Table 4-51 Clustering Histograms View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
BIN_ID	Bin ID
LOWER_BIN_BOUNDARY	Numeric lower bin boundary
UPPER_BIN_BOUNDARY	Numeric upper bin boundary
ATTRIBUTE_VALUE	Categorical attribute value
COUNT	Histogram count

The rule view DM$VR*model_name* describes the rule level information about a clustering model. The information is provided at attribute predicate level. The view has the following columns:

```
Name                           Type
------------------------------ ---------------------------
PARTITION_NAME                 VARCHAR2(128)
CLUSTER_ID                     NUMBER
CLUSTER_NAME                   NUMBER/VARCHAR2
ATTRIBUTE_NAME                 VARCHAR2(128)
ATTRIBUTE_SUBNAME              VARCHAR2(4000)
OPERATOR                       VARCHAR2(2)
NUMERIC_VALUE                  NUMBER
ATTRIBUTE_VALUE                VARCHAR2(4000)
SUPPORT                        NUMBER
CONFIDENCE                     BINARY_DOUBLE
RULE_SUPPORT                   NUMBER
RULE_CONFIDENCE                BINARY_DOUBLE
```

Table 4-52 Clustering Rules View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster

Table 4-52 (Cont.) Clustering Rules View

Column Name	Description
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
OPERATOR	Attribute predicate operator - a conditional operator taking the following values: *IN*, = , <>, < , >, <=, and >=
NUMERIC_VALUE	Numeric lower bin boundary
ATTRIBUTE_VALUE	Categorical attribute value
SUPPORT	Attribute predicate support
CONFIDENCE	Attribute predicate confidence
RULE_SUPPORT	Rule level support
RULE_CONFIDENCE	Rule level confidence

4.9.16 Model Detail Views for Expectation Maximization

Model detail views specific to Expectation Maximization (EM) contain additional information about an EM model. Additional views are available for EM Clustering, but are absent for EM Anomaly.

These are the model views available for Expectation Maximization:

Model Views	Description
DM$VA*model_name*	Clustering Attribute Statistics
DM$VB*model_name*	Attribute Pair Kullback-Leibler Divergence
DM$VD*model_name*	Clustering Description
DM$VF*model_name*	Expectation Maximization Bernoulli parameters
DM$VG*model_name*	Global Name-Value Pairs
DM$VH*model_name*	Clustering Histograms
DM$VI*model_name*	Unsupervised Attribute Importance
DM$VM*model_name*	Expectation Maximization Gaussian parameters
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VO*model_name*	Expectation Maximization Components
DM$VP*model_name*	Expectation Maximization Projections
DM$VR*model_name*	Clustering Rules
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

For EM Clustering model, the following views contain information that is not in the clustering views. For the clustering views, refer to "Model Detail Views for Clustering Algorithms".

The Expectation Maximization Components view (DM$VO*model_name*) describes the EM Cluster components. The component view contains information about their prior probabilities and what cluster they map to. The view has the following columns:

```
Name                                    Type
--------------------------------------- ----------------------------
```

```
PARTITION_NAME                   VARCHAR2(128)
COMPONENT_ID                     NUMBER
CLUSTER_ID                       NUMBER
PRIOR_PROBABILITY                BINARY_DOUBLE
```

Table 4-53 Expectation Maximization Components View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
COMPONENT_ID	Unique identifier of a component
CLUSTER_ID	The ID of a cluster in the model
PRIOR_PROBABILITY	Component prior probability

The Expectation Maximization Gaussian view (DM$VM*model_name*) provides information about the mean and variance parameters for the attributes by Gaussian distribution models. The view has the following columns:

```
Name                             Type
-------------------------------- ----------------------------
PARTITION_NAME                   VARCHAR2(128)
COMPONENT_ID                     NUMBER
ATTRIBUTE_NAME                   VARCHAR2(4000)
MEAN                             BINARY_DOUBLE
VARIANCE                         BINARY_DOUBLE
```

The Expectation Maximization Bernoulli parameters view (DM$VF*model_name*) provides information about the parameters of the multi-valued Bernoulli distributions used by the EM model. The view has the following columns:

```
Name                              Type
 -------------------------------- ----------------------------
 PARTITION_NAME                   VARCHAR2(128)
 COMPONENT_ID                     NUMBER
 ATTRIBUTE_NAME                   VARCHAR2(4000)
 ATTRIBUTE_VALUE                  VARCHAR2(4000)
 FREQUENCY                        BINARY_DOUBLE
```

Table 4-54 Expectation Maximization Bernoulli parameters View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
COMPONENT_ID	Unique identifier of a component
ATTRIBUTE_NAME	Column name
ATTRIBUTE_VALUE	Categorical attribute value
FREQUENCY	The frequency of the multivalued Bernoulli distribution for the attribute/value combination specified by ATTRIBUTE_NAME and ATTRIBUTE_VALUE.

For 2-Dimensional columns, EM provides an attribute ranking similar to that of attribute importance. This ranking is based on a rank-weighted average over Kullback–Leibler divergence computed for pairs of columns. This unsupervised attribute importance is shown in the Unsupervised Attribute Importance view (DM$VI*model_name*) and has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
ATTRIBUTE_NAME                            VARCHAR2(128)
ATTRIBUTE_IMPORTANCE_VALUE                BINARY_DOUBLE
ATTRIBUTE_RANK                            NUMBER
```

Table 4-55 Unsupervised Attribute Importance View for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_IMPORTANCE_VALUE	Importance value
ATTRIBUTE_RANK	An attribute rank based on the importance value

The pairwise Kullback–Leibler divergence is reported in the Attribute Pair Kullback-Leibler Divergence view (DM$VB*model_name*). This metric evaluates how much the observed joint distribution of two attributes diverges from the expected distribution under the assumption of independence. That is, the higher the value, the more dependent the two attributes are. The dependency value is scaled based on the size of the grid used for each pairwise computation. That ensures that all values fall within the [0; 1] range and are comparable. The view has the following columns:

```
Name                                      Type
----------------------------------------- ----------------------------
PARTITION_NAME                            VARCHAR2(128)
ATTRIBUTE_NAME_1                          VARCHAR2(128)
ATTRIBUTE_NAME_2                          VARCHAR2(128)
DEPENDENCY                                BINARY_DOUBLE
```

Table 4-56 Attribute Pair Kullback-Leibler Divergence View for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME_1	Name of the first attribute
ATTRIBUTE_NAME_2	Name of the second attribute
DEPENDENCY	Scaled pairwise Kullback-Leibler divergence

The projection table DM$VP*model_name* shows the coefficients used by random projections to map nested columns to a lower dimensional space. The view has rows only when nested or text data is present in the build data. The view has the following columns:

```
Name                             Type
-------------------------------- ----------------------------
PARTITION_NAME                   VARCHAR2(128)
FEATURE_NAME                     VARCHAR2(4000)
ATTRIBUTE_NAME                   VARCHAR2(128)
ATTRIBUTE_SUBNAME                VARCHAR2(4000)
ATTRIBUTE_VALUE                  VARCHAR2(4000)
COEFFICIENT                      NUMBER
```

Table 4-57 Projection table for Expectation Maximization

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_NAME	Name of feature
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	Projection coefficient. The representation is sparse; only the non-zero coefficients are returned.

For EM Anomaly, currently there are no additional views other than the classification views. For the classification view, refer to “Model Detail Views for Classification Algorithms”.

Global Details for Expectation Maximization

The following table describes global details for EM.

Table 4-58 Global Details for Expectation Maximization

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The possible values are: • YES • NO
LOGLIKELIHOOD	Loglikelihood on the build data
NUM_COMPONENTS	Number of components produced by the model
NUM_CLUSTERS	Number of clusters produced by the model (only available for EM Clustering)
NUM_ROWS	Number of rows used in the build
RANDOM_SEED	The random seed value used for the model build
REMOVED_COMPONENTS	The number of empty components excluded from the model

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

4.9.17 Model Detail Views for *k*-Means

Model detail views specific to *k*-Means (KM) contain clustering description view (`DM$VG`), and scoring information.

The following model views are available for *k*-Means algorithm.

Model Views	Description
`DM$VA`*model_name*	Clustering Attribute Statistics
`DM$VC`*model_name*	k-Means Scoring Centroids
`DM$VD`*model_name*	Clustering Description
`DM$VG`*model_name*	Global Name-Value Pairs
`DM$VH`*model_name*	Clustering Histograms
`DM$VN`*model_name*	Normalization and Missing Value Handling
`DM$VR`*model_name*	Clustering Rules
`DM$VS`*model_name*	Computed Settings
`DM$VW`*model_name*	Model Build Alerts

"Model Detail Views for Clustering Algorithms" discusses common model views across clustering algorithms. Global Name-Value Pairs view (`DM$VG`), which contains information about Computed Settings view (`DM$VS`) and Model Build Alerts view (`DM$VW`), and Normalization and Missing Value Handling view (`DM$VN`) are addressed individually.

The following views contain information that is specific to *k*-Means model.

The *k*-Means Clustering Description view `DM$VD`*model_name* has an additional column:

```
Name                              Type
--------------------------------- ----------------------------
DISPERSION                        BINARY_DOUBLE
```

Table 4-59 Clustering Description for k-Means

Column Name	Description
`DISPERSION`	A measure used to quantify whether a set of observed occurrences are dispersed compared to a standard statistical model.

The *k*-Means Scoring Centroids view `DM$VC`*model_name* describes the centroid of each leaf clusters:

```
Name                              Type
 --------------------------------- ----------------------------
 PARTITION_NAME                    VARCHAR2(128)
 CLUSTER_ID                        NUMBER
 CLUSTER_NAME                      NUMBER/VARCHAR2
```

ATTRIBUTE_NAME	VARCHAR2(128)
ATTRIBUTE_SUBNAME	VARCHAR2(4000)
ATTRIBUTE_VALUE	VARCHAR2(4000)
VALUE	BINARY_DOUBLE

Table 4-60 k-Means Scoring Centroids View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CLUSTER_ID	The ID of a cluster in the model
CLUSTER_NAME	Specifies the label of the cluster
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
VALUE	Specifies the centroid value

The following table describes Global Name-Value Pairs view (DM$VG) for *k*-Means.

Table 4-61 *k*–Means Global Name-Value Pairs View

Name	Description
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
NUM_ROWS	Number of rows used in the build
REMOVED_ROWS_ZERO_NORM	Number of rows removed due to 0 norm. This applies only to models using cosine distance.

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).
- Model Detail Views for Global Information
  Model detail views for global information contain information about global statistics, alerts, and computed settings.

4.9.18 Model Detail Views for O-Cluster

Model detail views specific to O-Cluster (OC) contain information about description view, histograms view, and global view.

These are the available model views for O-Cluster:

Model Views	Description
DM$VA*model_name*	Clustering Attribute Statistics

Model Views	Description
DM$VB*model_name*	Automatic Data Preparation Binning
DM$VD*model_name*	Clustering Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VH*model_name*	Clustering Histograms
DM$VR*model_name*	Clustering Rules
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The following views contain information that is specific to an O-Cluster model. For the clustering views, refer to "Model Detail Views for Clustering Algorithms". The OC algorithm uses the same descriptive statistics views as Expectation Maximization (EM) and *k*-Means (KM). The following are the statistics views:

The Cluster Description view (DM$VD*model_name*) describes the O-Cluster components. The Cluster Description view has additional fields that specify the split predicate. The view has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
ATTRIBUTE_NAME                    VARCHAR2(128)
ATTRIBUTE_SUBNAME                 VARCHAR2(4000)
OPERATOR                          VARCHAR2(2)
VALUE                             SYS.XMLTYPE
```

Table 4-62 Cluster Description View for O-Cluster

Column Name	Description
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
OPERATOR	Split operator
VALUE	List of split values

The structure of the SYS.XMLTYPE is as follows:

```
<Element>splitval1</Element>
```

The OC algorithm uses a Clustering Histograms view (DM$VH*model_name*) with different columns than EM and KM. The view has the following columns:

```
Name                              Type
--------------------------------- ----------------------------
PARTITON_NAME                     VARCHAR2(128)
CLUSTER_ID                        NUMBER
ATTRIBUTE_NAME                    VARCHAR2(128)
ATTRIBUTE_SUBNAME                 VARCHAR2(4000)
BIN_ID                            NUMBER
```

LABEL	VARCHAR2(4000)
COUNT	NUMBER

Table 4-63 Clustering Histograms View for O-Cluster

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CLUSTER_ID	Unique identifier of a component
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
BIN_ID	Unique identifier
LABEL	Bin label
COUNT	Bin histogram count

The following table describes the Global Name-Value Pairs (DM$VG*model_name*) view specific to O-Cluster.

Table 4-64 O-Cluster Statistics Information In Model Global View

Name	Description
NUM_ROWS	The total number of rows used in the build

Related Topics

- Model Detail Views for Clustering Algorithms
  Oracle Machine Learning for SQL supports these clustering algorithms: Expectation Maximization (EM), *k*-Means (KM), and orthogonal partitioning clustering (O-Cluster, OC).

4.9.19 Model Detail Views for Explicit Semantic Analysis

Model detail views specific to Explicit Semantic Analysis (ESA) contain information about attribute statistics and features.

These are the available model views:

Model Views	Description
DM$VA*model_name*	Explicit Semantic Analysis Matrix
DM$VF*model_name*	Explicit Semantic Analysis Features
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts
DM$VX*model_name*	Text Features

- Explicit Semantic Analysis Matrix (DM$VA*model_name*): This view has different columns for feature extraction and classification. For feature extraction, this view contains model attribute coefficients per feature. For classification, this view contains model attribute coefficients per target class.

- Explicit Semantic Analysis Features (DM$VF*model_name*): This view is applicable only for feature extraction.

The Explicit Semantic Analysis Matrix view (DM$VA*model_name*) has the following columns for feature extraction:

```
Name                             Type
-------------------------------- --------------------------
PARTITION_NAME                   VARCHAR2(128)
FEATURE_ID                       NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                 TIMESTAMP WITH TIME ZONE,
                                 TIMESTAMP WITH LOCAL TIME ZONE
ATTRIBUTE_NAME                   VARCHAR2(128)
ATTRIBUTE_SUBNAME                VARCHAR2(4000)
ATTRIBUTE_VALUE                  VARCHAR2(4000)
COEFFICIENT                      BINARY_DOUBLE
```

Table 4-65 Explicit Semantic Analysis Matrix for Feature Extraction

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	Unique identifier of a feature as it appears in the training data
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	A measure of the weight of the attribute with respect to the feature

The (DM$VA*model_name*) view comprises of attribute coefficients for all target classes.

The view Explicit Semantic Analysis Matrix (DM$VA*model_name*) has the following columns for classification:

```
Name                             Type
-------------------------------- --------------------------
PARTITION_NAME                   VARCHAR2(128)
TARGET_VALUE                     NUMBER/VARCHAR2
ATTRIBUTE_NAME                   VARCHAR2(128)
ATTRIBUTE_SUBNAME                VARCHAR2(4000)
ATTRIBUTE_VALUE                  VARCHAR2(4000)
COEFFICIENT                      BINARY_DOUBLE
```

Table 4-66 Explicit Semantic Analysis Matrix for Classification

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
TARGET_VALUE	Value of the target
ATTRIBUTE_NAME	Column name

Table 4-66 (Cont.) Explicit Semantic Analysis Matrix for Classification

Column Name	Description
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value
COEFFICIENT	A measure of the weight of the attribute with respect to the feature

The Explicit Semantic Analysis Features view (DM$VF*model_name*) has a unique row for every feature in one view. This feature is helpful if the model was pre-built and the source training data are not available. The view has the following columns:

```
Name                              Type
--------------------------------- ---------------------------
PARTITION_NAME                    VARCHAR2(128)
FEATURE_ID                        NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                  TIMESTAMP WITH TIME ZONE,
                                  TIMESTAMP WITH LOCAL TIME ZONE
```

Table 4-67 Explicit Semantic Analysis Features for Explicit Semantic Analysis

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	Unique identifier of a feature as it appears in the training data

The following table describes the Global Name-Value Pairs view (DM$VG*model_name)* specific to ESA.

Table 4-68 Explicit Semantic Analysis Statistics Information In Model Global View

Name	Description
NUM_ROWS	The total number of input rows
REMOVED_ROWS_BY_FILTERS	Number of rows removed by filters

4.9.20 Model Detail Views for Non-Negative Matrix Factorization

Model detail views specific to Non-Negative Matrix Factorization (NMF) contain information about the encoding H matrix and H inverse matrix.

These are the available model views for NMF:

Model Views	Description
DM$VE*model_name*	Non-Negative Matrix Factorization H Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VI*model_name*	Non-Negative Matrix Factorization Inverse H Matrix
DM$VN*model_name*	Normalization andMissing Value Handling

Model Views	Description
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The views specific to NMF are:

- Non-Negative Matrix Factorization H Matrix view (DM$VE*model_name*)
- Non-Negative Matrix Factorization Inverse H Matrix view (DM$VI*model_name*)

The view DM$VE*model_name* describes the encoding (H) matrix of an NMF model. The FEATURE_NAME column type may be either NUMBER or VARCHAR2. The view has the following columns.

```
Name                 Type
-------------------  -------------------------
PARTITION_NAME       VARCHAR2(128)
FEATURE_ID           NUMBER
FEATURE_NAME         NUMBER/VARCHAR2
ATTRIBUTE_NAME       VARCHAR2(128)
ATTRIBUTE_SUBNAME    VARCHAR2(4000)
ATTRIBUTE_VALUE      VARCHAR2(4000)
COEFFICIENT          BINARY_DOUBLE
```

Table 4-69 Non-Negative Matrix Factorization H Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Specifies the value of attribute
COEFFICIENT	The attribute encoding that represents its contribution to the feature

The view DM$VI*model_view* describes the inverse H matrix of an NMF model. The FEATURE_NAME column type may be either NUMBER or VARCHAR2. The view has the following schema:

```
Name                 Type
-------------------  -------------------------
PARTITION_NAME       VARCHAR2(128)
FEATURE_ID           NUMBER
FEATURE_NAME         NUMBER/VARCHAR2
ATTRIBUTE_NAME       VARCHAR2(128)
ATTRIBUTE_SUBNAME    VARCHAR2(4000)
ATTRIBUTE_VALUE      VARCHAR2(4000)
COEFFICIENT          BINARY_DOUBLE
```

Table 4-70 Non-Negative Matrix Factorization Inverse H Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Specifies the value of attribute
COEFFICIENT	The attribute encoding that represents its contribution to the feature

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to NMF.

Table 4-71 Global Name-Value Pairs View for NMF

Name	Description
CONV_ERROR	Convergence error
CONVERGED	Indicates whether the model build process has converged to specified tolerance. The following are the possible values: • YES • NO
ITERATIONS	Number of iterations performed during build
NUM_ROWS	Number of rows used in the build input data set
SAMPLE_SIZE	Number of rows used by the build

4.9.21 Model Detail Views for Singular Value Decomposition

Model detail views specific to Singular Value Decomposition (SVD) contain information about the S matrix, right-singular vectors, and left-singular vectors.

These are the available model views for SVD:

Model Views	Description
DM$VE*model_name*	Singular Value Decomposition S Matrix
DM$VG*model_name*	Global Name-Value Pairs
DM$VN*model_name*	Normalization and Missing Value Handling
DM$VS*model_name*	Computed Settings
DM$VU*model_name*	Singular Value Decomposition U Matrix
DM$VV*model_name*	Singular Value Decomposition V Matrix
DM$VW*model_name*	Model Build Alerts

The Singular Value Decomposition S Matrix view (DM$VE*model_name*) leverages the fact that each singular value in the SVD model has a corresponding principal component in the

associated Principal Components Analysis (PCA) model to relate a common set of information for both classes of models. For an SVD model, it describes the content of the S matrix. When PCA scoring is selected as a build setting, the variance and percentage cumulative variance for the corresponding principal components are shown as well. The view has the following columns:

```
Name                             Type
-------------------------------- ---------------------------
PARTITION_NAME                   VARCHAR2(128)
FEATURE_ID                       NUMBER
FEATURE_NAME                     NUMBER/VARCHAR2
VALUE                            BINARY_DOUBLE
VARIANCE                         BINARY_DOUBLE
PCT_CUM_VARIANCE                 BINARY_DOUBLE
```

Table 4-72 Singular Value Decomposition S Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
VALUE	The matrix entry value
VARIANCE	The variance explained by a component. This column is only present for SVD models with setting dbms_data_mining.svds_scoring_mode set to dbms_data_mining.svds_scoring_pca This column is non-null only if the build data is centered, either manually or because of the following setting:dbms_data_mining.prep_auto is set to dbms_data_mining.prep_auto_on.
PCT_CUM_VARIANCE	The percent cumulative variance explained by the components thus far. The components are ranked by the explained variance in descending order. This column is only present for SVD models with setting dbms_data_mining.svds_scoring_mode set to dbms_data_mining.svds_scoring_pca This column is non-null only if the build data is centered, either manually or because of the following setting:dbms_data_mining.prep_auto is set to dbms_data_mining.prep_auto_on.

The Singular Value Decomposition V Matrix view (DM$VV*model_view*) describes the right-singular vectors of an SVD model. For a PCA model it describes the principal components (eigenvectors). The view has the following columns:

```
Name                             Type
-------------------------------- ---------------------------
PARTITION_NAME                   VARCHAR2(128)
FEATURE_ID                       NUMBER
FEATURE_NAME                     NUMBER/VARCHAR2
ATTRIBUTE_NAME                   VARCHAR2(128)
ATTRIBUTE_SUBNAME                VARCHAR2(4000)
```

```
ATTRIBUTE_VALUE                  VARCHAR2(4000)
VALUE                            BINARY_DOUBLE
```

Table 4-73 Singular Value Decomposition V Matrix View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_VALUE	Categorical attribute value. For numerical attributes, ATTRIBUTE_VALUE is null.
VALUE	The matrix entry value

The Singular Value Decomposition U Matrix view (DM$VU*model_name*) describes the left-singular vectors of an SVD model. For a PCA model, it describes the projection of the data in the principal components. This view does not exist unless the settings dbms_data_mining.svds_u_matrix_output is set to dbms_data_mining.svds_u_matrix_enable. The view has the following columns:

```
Name                             Type
-------------------------------- ---------------------------
PARTITION_NAME                   VARCHAR2(128)
CASE_ID                          NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                 TIMESTAMP WITH TIME ZONE,
                                 TIMESTAMP WITH LOCAL TIME ZONE
FEATURE_ID                       NUMBER
FEATURE_NAME                     NUMBER/VARCHAR2
VALUE                            BINARY_DOUBLE
```

Table 4-74 Singular Value Decomposition U Matrix View or Projection Data in Principal Components

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
CASE_ID	Unique identifier of the row in the build data described by the **U** matrix projection.
FEATURE_ID	The ID of a feature in the model
FEATURE_NAME	The name of a feature in the model
VALUE	The matrix entry value

Global Details for Singular Value Decomposition

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to a SVD model.

Table 4-75 Global Name-Value Pairs View for Singular Value Decomposition

Name	Description
NUM_COMPONENTS	Number of features (components) produced by the model
NUM_ROWS	The total number of rows used in the build
SUGGESTED_CUTOFF	Suggested cutoff that indicates how many of the top computed features capture most of the variance in the model. Using only the features below this cutoff would be a reasonable strategy for dimensionality reduction.

Related Topics

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

4.9.22 Model Detail Views for Minimum Description Length

Model detail views specific to Minimum Description Length (MDL) (for calculating attribute importance) contain information about attribute importance models.

These are the available model views for MDL:

Model Views	Description
DM$VA*model_name*	Attribute Importance
DM$VB*model_name*	Automatic Data Preparation Binning
DM$VG*model_name*	Global Name-Value Pairs
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts

The Attribute Importance view (DM$VA*model_name*) describes the attribute importance as well as the attribute importance rank. The view has the following columns:

```
Name                                    Type
--------------------------------------- --------------------------
PARTITION_NAME                          VARCHAR2(128)
ATTRIBUTE_NAME                          VARCHAR2(128)
ATTRIBUTE_SUBNAME                       VARCHAR2(4000)
ATTRIBUTE_IMPORTANCE_VALUE              BINARY_DOUBLE
ATTRIBUTE_RANK                          NUMBER
```

Table 4-76 Attribute Importance View for Minimum Description Length

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
ATTRIBUTE_IMPORTANCE_VALUE	Importance value
ATTRIBUTE_RANK	Rank based on importance

The following table describes the Global Name-Value Pairs view (DM$VG*model_name*) specific to MDL.

Table 4-77 Global Name-Value Pairs View for MDL

Name	Description
NUM_ROWS	The total number of rows used in the build

4.9.23 Model Detail Views for Binning

The binning view DM$VB describes the bin boundaries used in automatic data preparation.

The view has the following columns:

```
Name                    Type
--------------------    --------------------
PARTITION_NAME          VARCHAR2(128)
ATTRIBUTE_NAME          VARCHAR2(128)
ATTRIBUTE_SUBNAME       VARCHAR2(4000)
BIN_ID                  NUMBER
LOWER_BIN_BOUNDARY      BINARY_DOUBLE
UPPER_BIN_BOUNDARY      BINARY_DOUBLE
ATTRIBUTE_VALUE         VARCHAR2(4000)
```

Table 4-78 Model Details View for Binning

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ATTRIBUTE_NAME	Specifies the attribute name
ATTRIBUTE_SUBNAME	Specifies the attribute subname
BIN_ID	Bin ID (or bin identifier)
LOWER_BIN_BOUNDARY	Numeric lower bin boundary
UPPER_BIN_BOUNDARY	Numeric upper bin boundary
ATTRIBUTE_VALUE	Categorical value

4.9.24 Model Detail Views for Global Information

Model detail views for global information contain information about global statistics, alerts, and computed settings.

The Global Name-Value Pairs view (DM$VG*model_name*) describes global statistics related to the model build. Examples include the number of rows used in the build, the convergence status, and the model quality metrics. The view has the following columns:

```
Name                    Type
--------------------    --------------------
PARTITION_NAME          VARCHAR2(128)
NAME                    VARCHAR2(30)
NUMERIC_VALUE           NUMBER
STRING_VALUE            VARCHAR2(4000)
```

Table 4-79 Global Name-Value Pairs View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
NAME	Name of the statistic
NUMERIC_VALUE	Numeric value of the statistic
STRING_VALUE	Categorical value of the statistic

The Model Build Alerts view (DM$VW*model_name*) lists alerts issued during the model build. The view has the following columns:

```
Name                 Type
-------------------  ---------------------
PARTITION_NAME       VARCHAR2(128)
ERROR_NUMBER         BINARY_DOUBLE
ERROR_TEXT           VARCHAR2(4000)
```

Table 4-80 Model Build Alerts View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
ERROR_NUMBER	Error number (valid when event is Error)
ERROR_TEXT	Error message

The Computed Settings view (DM$VS*model_name*) lists the algorithm computed settings. The view has the following columns:

```
Name                 Type
-----------------    --------------------
PARTITION_NAME       VARCHAR2(128)
SETTING_NAME         VARCHAR2(30)
SETTING_VALUE        VARCHAR2(4000)
```

Table 4-81 Computed Settings View

Column Name	Description
PARTITION_NAME	Partition name in a partitioned model
SETTING_NAME	Name of the setting
SETTING_VALUE	Value of the setting

4.9.25 Model Detail Views for Normalization and Missing Value Handling

The Normalization and Missing Value Handling view DM$VN describes the normalization parameters used in Automatic Data Preparation (ADP) and the missing value replacement

when a NULL value is encountered. Missing value replacement applies only to the two-dimensional columns and does not apply to the nested columns.

The view has the following columns:

```
Name                          Type
----------------------        ----------------------
PARTITION_NAME                VARCHAR2(128)
ATTRIBUTE_NAME                VARCHAR2(128)
ATTRIBUTE_SUBNAME             VARCHAR2(4000)
NUMERIC_MISSING_VALUE         BINARY_DOUBLE
CATEGORICAL_MISSING_VALUE     VARCHAR2(4000)
NORMALIZATION_SHIFT           BINARY_DOUBLE
NORMALIZATION_SCALE           BINARY_DOUBLE
```

Table 4-82 Normalization and Missing Value Handling View

Column Name	Description
PARTITION_NAME	A partition in a partitioned model
ATTRIBUTE_NAME	Column name
ATTRIBUTE_SUBNAME	Nested column subname. The value is null for non-nested columns.
NUMERIC_MISSING_VALUE	Numeric missing value replacement
CATEGORICAL_MISSING_VALUE	Categorical missing value replacement
NORMALIZATION_SHIFT	Normalization shift value
NORMALIZATION_SCALE	Normalization scale value

4.9.26 Model Detail Views for Exponential Smoothing

Model detail views specific to Exponential Smoothing (ESM) include information about the model output, global information about the model, and views that support time series regression.

These are the available model views for ESM:

Model Details	Description
DM$VG*model_name*	Global Name-Value Pairs
DM$VP*model_name*	Exponential Smoothing Forecast
DM$VS*model_name*	Computed Settings
DM$VW*model_name*	Model Build Alerts
DM$VR*model_name*	Time Series Regression Build
DM$VT*model_name*	Time Series Regression Score

Exponential Smoothing Forecast view (DM$VP*model_name*) displays the outcome of an ESM model. The output contains a set of records, ordered by partition and CASE_ID, that include the columns given in the *Exponential Smoothing Model Output* table. CASE_ID identifies the value's position in the time series. The user-specified CASE_ID can be a type that represents a numerical or datetime value. For each unique value of PARTITION, a distinct exponential smoothing model is built. The VALUE column for each PARTITION represents the observed or accumulated value of the target at that point in the sequence. The PREDICTION column is the

forecast one step ahead at that point in the sequence. Backcasts are predictions that fall inside the range of the input data. The sequence also includes a user-specified number of values beyond the range of the input data. The VALUE column is *NULL* for any sequence value outside the range of input, and PREDICTION column is the model forecast for that sequence value. Lower and upper boundaries of the forecasts are denoted by the LOWER and UPPER columns. For backcasts, LOWER and UPPER are *NULL*. The bounds are based on a confidence interval that the user sets for the prediction.

Table 4-83 Exponential Smoothing Forecast View

Name	Description
PARTITION	Partition name in a partitioned model
CASE_ID	Sequence identifier (datetime or number type)
VALUE	Observed or accumulated value
PREDICTION	Backcast or Forecast value
UPPER	Upper bound of the forecast
LOWER	Lower bound of the forecast

Global Name-Value Pairs view (DM$VG*model_name*) includes the model's global information as well as the estimated smoothing constants, estimated initial state, and global diagnostic measures.

Depending on the type of model, the global diagnostics include some or all of the following for Exponential Smoothing.

Table 4-84 Global Name-Value Pairs View for ESM

Name	Description
-2 LOG-LIKELIHOOD	Negative log-likelihood of model
ALPHA	Smoothing constant
AIC	Akaike information criterion
AICC	Corrected Akaike information criterion
AMSE	Average mean square error over user-specified time window
BETA	Trend smoothing constant
BIC	Bayesian information criterion
GAMMA	Seasonal smoothing constant
INITIAL LEVEL	Model estimate of value one time interval prior to start of observed series
INITIAL SEASON i	Model estimate of seasonal effect for season *i* one time interval prior to start of observed series
INITIAL TREND	Model estimate of trend one time interval prior to start of observed series
MAE	Model mean absolute error
MSE	Model mean square error
PHI	Damping parameter

Table 4-84 (Cont.) Global Name-Value Pairs View for ESM

Name	Description
STD	Model standard error
SIGMA	Model standard deviation of residuals

Time series regression expands the features that can be included in a time series model and, possibly, increases forecast accuracy. Backcasts and forecasts of time series correlated to the "target" series of interest are included in the build and score views. The build and score views can be fed into a regression technique like Generalized Linear Model.

The Time Series Regression Build view (DM$VR*model_name*) depicts the schema for the build view. Each predictor series will have its own column. There can be a maximum of 20 predictor series in the build and score views. The names of the columns are obtained from the EXSM_SERIES_LIST setting.

Table 4-85 Time Series Regression Build View

Name	Description
PARTITION	Partition name in a partitioned model
CASE_ID	Sequence identifier (datetime or number type)
target series name	Observed or accumulated value of target series
DM$*target series*	Backcasted value of target series
DM$*predictor series column name*	Backcasted value of predictor series column. A maximum of *20* predictor series columns can be used.

The Time Series Regression Score view (DM$VT*model_name*) shows the schema for the score view. The schema is the same as in the build view, but the values in the *target series name* column are NULL because the future has not yet been observed.

Table 4-86 Time Series Regression Score View

Name	Description
PARTITION	Partition name in a partitioned model
CASE_ID	Sequence identifier (datetime or number type)
target series name	NULLs, because the future values of the target series have not been observed
DM$*target series*	Forecasted value of target series
DM$*predictor series column name*	Forecasted value of predictor series column name. A maximum of *20* predictor series columns can be used.

Related Topics

- About Exponential Smoothing
- About Generalized Linear Models

4.9.27 Model Detail Views for Text Features

The model details view for text features is DM$VX*model_name*.

The text feature view DM$VX*model_name* describes the extracted text features if there are text attributes present. The view has the following schema:

```
Name                                      Type
 --------------                    --------------------
 PARTITION_NAME                            VARCHAR2(128)
 COLUMN_NAME                               VARCHAR2(128)
 TOKEN                                     VARCHAR2(4000)
 DOCUMENT_FREQUENCY                        NUMBER
```

Table 4-87 Text Feature View for Extracted Text Features

Column Name	Description
PARTITION_NAME	A partition in a partitioned model to retrieve details
COLUMN_NAME	Name of the identifier column
TOKEN	Text token which is usually a word or stemmed word
DOCUMENT_FREQUENCY	A measure of token frequency in the entire training set

4.9.28 Model Detail Views for ONNX Models

You can view the details of an embedding model using the model detail views. The names of the views begin with DM$V.

This section lists the model detail views for embedding models.

- DM$VJ Model Detail View
  The DM$VJ<*model-name*> returns a single row containing a JSON object in one column that contains user-specified metadata of the model.
- DM$VM Model Detail View
  The DM$VM<*model-name*> view reports information extracted from the metadata of the imported ONNX model and its input or output tensors.
- DM$VP Model Detail View
  The DM$VP<*model-name*> view displays information extracted from parsing the JSON metadata. The view presents the JSON metadata of the model, including both explicitly declared properties and system-assigned default values for undeclared ones.
- DM$VG Model Detail View
  The DM$VG<*model-name*> displays information of all the external data groups.
- DM$VX Model Detail View
  The DM$VX<*model-name*> displays metadata for each external data.

4.9.28.1 DM$VJ Model Detail View

The DM$VJ<*model-name*> returns a single row containing a JSON object in one column that contains user-specified metadata of the model.

The view has the following columns:

```
Name                          Null?    Type
----------------------------- -------- ----------------------------
 METADATA                              CLOB
```

Column Name	Description
METADATA	It is a CLOB containing the user-specified metadata of the embedding model in JSON format.

The following table describes the output of the DM$VJ<*modle_name*> view of an embedding model.

Name	Value
METADATA	The JSON that was specified to the IMPORT_ONNX_MODEL call for importing the model.

The following example displays the output of an embedding model. The name of the model is *doc_model*:

```
select * from DM$VJdoc_model;
```

The output is as follows:

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

4.9.28.2 DM$VM Model Detail View

The DM$VM<*model-name*> view reports information extracted from the metadata of the imported ONNX model and its input or output tensors.

The view has the following columns:

```
Name                                Type
----------------------------------- --------------------------
NAME                               VARCHAR2(4000)
VALUE                                 VARCHAR2(4000)
```

Table 4-88

Column Name	Description
NAME	The name of the metadata extracted from the ONNX model.
VALUE	Indicates a value for the metadata name

The following table describes the output of the DM$VM<*model_name*> view of an embedding model.

Name	Value
Producer Name	Name of the tools that generated the ONNX files
Graph Name	Name of the ONNX graph
Graph Description	Description given to the model
Version	Version of the model
Input	Describes the model input mapping
Output	Reports the vector information with dimension and value type

The following example displays the output of an embedding model. The name of the model is *DOC_MODEL*:

```
select * from DM$VMdoc_model;
```

> **Note**
>
> If the model is an embedding model with external initializers produced by OML4Py, you can identify the producer by inspecting the output of this view DM$VMdoc_model. This view includes the Producer Name field (if populated during model import) that indicate the tool or framework used to generate the model. The default is OML4Py.

The following is the output:

```
NAME             VALUE
-------------------- ----------------------------------------
Producer Name         OML4Py
Graph Name          tokenizer_main_graph
Graph Description    Graph combining tokenizer and main_graph

             tokenizer



             main_graph


Version          1
Input[0]         input:string[1]
Output[0]         embedding:float32[?,384]

6 rows selected.
```

Related Topics

- https://github.com/onnx/onnx/blob/main/docs/IR.md

4.9.28.3 DM$VP Model Detail View

The DM$VP<*model-name*> view displays information extracted from parsing the JSON metadata. The view presents the JSON metadata of the model, including both explicitly declared properties and system-assigned default values for undeclared ones.

The reported properties are specific to the machine learning model and match the mandatory and optional fields of the JSON metadata.

The view has the following columns:

```
Name                                      Type
----------------------------------- --------------------------
NAME                                VARCHAR2(4000)
VALUE                                  VARCHAR2(4000)
```

Column Name	Description
NAME	Displays the JSON parameters
VALUE	Indicates the value corresponding to the JSON parameter name value pair

Note that this information is already available in the ALL_MINING_MODEL_ATTRIBUTES view. The following example displays all the columns available to you in the DM$VP*doc_model* view of an embedding model. In this example, *doc_model* is the name of the model.

```
select * from DM$VPdoc_model;
```

The output is as follows:

```
NAME                                     VALUE
---------------------------------------
---------------------------------------
batching                                 False
embeddingOutput                          embedding
```

4.9.28.4 DM$VG Model Detail View

The DM$VG<*model-name*> displays information of all the external data groups.

The view has the following columns:

Column Name	Description
NAME	Specifies the name assigned to the group of external data for storing tensor initializers.
NUM	Indicates the total number of tensors contained within the external data group.
SIZE	Specifies the size, in bytes, of the external data group.
METADATA	Provides metadata associated with the external data group.

The following example displays the output of a model with external data group. The name of the model is *DOC_MODEL*:

```
select * from DM$VGDOC_MODEL;
```

The output is as follows:

```
select * from DM$VGDOC_MODEL;

NAME                        NUM  DATA_SIZE METADATA
--------------------------- ---------- ----------
---------------------------------------
doc_model_one_group_0.data           197   33397504
{"group_name":"doc_model_one_group_0.data",
                             "tensors":[{"name":"_model.embeddings.La
```

> **Note**
>
> The METADATA column can be very large because an external data group may include hundreds of external initializers, and the column stores metadata for each initializer.

4.9.28.5 DM$VX Model Detail View

The DM$VX<*model-name*> displays metadata for each external data.

The view has the following columns:

Column Name	Description
NAME	Specifies the name of the initializer that must be linked to the associated external data.
TYPE	Specifies the ONNX data type for each element in the tensor used as an initializer value.
OFFSET	Indicates the byte offset to the starting position of the tensor within its associated external data group.
SIZE	Specifies the size of the tensor in bytes within its external data source.
SHAPE	Provides a string describing the shape of the tensor, truncated to 4000 characters.
GROUP_NAME	Specifies the name of the external data group that contains the tensor associated with this initializer.

The following example displays the content of the view for a model with external initializers. The query retrieves only the first five rows, as the view contains one row per external initializer, and models can have several hundred initializers. The name of the model is *DOC_MODEL*:

```
select * from DM$VXDOC_MODEL;
```

The output is as follows:

```
SQL> select * from DM$VXDOC_MODEL where rownum < 5;

NAME                    TYPE            OFFSET    DATA_SIZE SHAPE
GROUP_NAME
------------------------------ ------------------- ---------- ----------
---------- ------------------------------
onnx::MatMul_1822_quantized    INT8                 0       589824 [1536,384]
doc_model_one_group_0.data
_model.embeddings.LayerNorm.we FLOAT            589824         1536
[384]      doc_model_one_group_0.data
ight

_model.embeddings.LayerNorm.bi FLOAT            591360         1536
[384]      doc_model_one_group_0.data
as

_model.encoder.layer.0.attenti FLOAT            592896         1536
[384]      doc_model_one_group_0.data
on.self.query.bias
```

Here `NAME` is the name of the external initializers. The initializer is stored at offset 0 in `CLOB` holding the content of the file *doc_model_one_group_0.data*.

5

Scoring and Deployment

Explains the scoring and deployment features of Oracle Machine Learning for SQL.

- About Scoring and Deployment
  Scoring is the application of models to new data. In Oracle Machine Learning for SQL, scoring is performed by SQL language functions.
- Use the Oracle Machine Learning for SQL Functions
  Some of the benefits of using SQL functions for Oracle Machine Learning for SQL are listed.
- Prediction Details
  Prediction details are XML strings that provide information about the score.
- Real-Time Scoring
  You can perform real-time scoring by running a SQL query. An example shows a real-time query using `PREDICTION_PROBABILITY` function. Based on the result, a customer representative can offer a value card to the customer.
- Dynamic Scoring
  You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.
- Cost-Sensitive Decision Making
  Costs are user-specified numbers that bias classification. The algorithm uses positive numbers to penalize more expensive outcomes over less expensive outcomes. Higher numbers indicate higher costs.
- DBMS_DATA_MINING.APPLY
  The `APPLY` procedure in `DBMS_DATA_MINING` is a batch apply operation that writes the results of scoring directly to a table.

5.1 About Scoring and Deployment

Scoring is the application of models to new data. In Oracle Machine Learning for SQL, scoring is performed by SQL language functions.

Predictive functions perform classification, regression, or anomaly detection. Clustering functions assign rows to clusters. Feature extraction functions transform the input data to a set of higher order predictors. A scoring procedure is also available in the `DBMS_DATA_MINING` PL/SQL package.

Deployment refers to the use of models in a target environment. Once the models have been built, the challenges come in deploying them to obtain the best results, and in maintaining them within a production environment. Deployment can be any of the following:

- Scoring data either for batch or real-time results. Scores can include predictions, probabilities, rules, and other statistics.
- Extracting model details to produce reports. For example: clustering rules, decision tree rules, or attribute rankings from an Attribute Importance model.

- Extending the business intelligence infrastructure of a data warehouse by incorporating machine learning results in applications or operational systems.
- Moving a model from the database where it was built to the database where it used for scoring (export/import)

OML4SQL supports all of these deployment scenarios.

> **Note**
>
> OML4SQL scoring operations support parallel execution. When parallel execution is enabled, multiple CPU and I/O resources are applied to the execution of a single database operation.
>
> Parallel execution offers significant performance improvements, especially for operations that involve complex queries and large databases typically associated with decision support systems (DSS) and data warehouses.

Related Topics

- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Machine Learning for SQL Concepts*
- Export and Import Oracle Machine Learning for SQL Models
  You can export machine learning models to move models to a different Oracle AI Database instance, such as from a development database to a production database.

5.2 Use the Oracle Machine Learning for SQL Functions

Some of the benefits of using SQL functions for Oracle Machine Learning for SQL are listed.

The OML4SQL functions provide the following benefits:

- Models can be easily deployed within the context of existing SQL applications.
- Scoring operations take advantage of existing query execution functionality. This provides performance benefits.
- Scoring results are pipelined, enabling the rows to be processed without requiring materialization.

The machine learning functions produce a score for each row in the selection. The functions can apply a machine learning model schema object to compute the score, or they can score dynamically without a pre-defined model, as described in "Dynamic Scoring".

- Choose the Predictors
  You can select different attributes as predictors in a `PREDICTION` function through a `USING` clause.
- Single-Record Scoring
  You can score a single record which produces 0 and 1 to predict customers who are unlikely or likely to use an affinity card.

Related Topics

- Dynamic Scoring
  You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.

- Scoring Requirements
  Learn how scoring is done in Oracle Machine Learning for SQL.
- Oracle Machine Learning for SQL Scoring Functions
  Use OML4SQL functions score data. Functions can apply a machine learning model schema object to data or dynamically mine it with an analytic clause. SQL functions exist for all OML4SQL scoring algorithms.
- *Oracle Database SQL Language Reference*

5.2.1 Choose the Predictors

You can select different attributes as predictors in a `PREDICTION` function through a `USING` clause.

The OML4SQL functions support a `USING` clause that specifies which attributes to use for scoring. You can specify some or all of the attributes in the selection and you can specify expressions. The following examples all use the `PREDICTION` function to find the customers who are likely to use an affinity card, but each example uses a different set of predictors.

When predictor values are not in the training data, the models score categorical values that were not in the training data without error. A score is produced using the remaining predictors. This enables batch scoring that does not fail because of a single record with an invalid value. Also, in some algorithms, like k-Means or Gaussian SVM, a new value can change the prediction in a meaningful way, such as resulting in larger distances with the unknown value. Furthermore, additional columns that were not present for building may be present in the table or view provided for scoring, and only the columns matching the model signature are used. Also, scoring may be performed with fewer predictors than are listed in the model signature.

In the case of partitioned models, a `NULL` score is produced if the partition value is invalid. If the partition column value is omitted, an error message is returned.

The query in Example 5-1 uses all the predictors.

The query in Example 5-2 uses only gender, marital status, occupation, and income as predictors.

The query in Example 5-3 uses three attributes and an expression as predictors. The prediction is based on gender, marital status, occupation, and the assumption that all customers are in the highest income bracket.

Example 5-1 Using All Predictors

The dt_sh_clas_sample model is created by the `oml4sql-classification-decision-tree.sql` example.

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

The output is follows:

```
C        CNT    AVG_AGE
- ---------- ----------
```

```
F          25          38
M         213          43
```

Example 5-2 Using Some Predictors

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

The output is as follows:

```
C         CNT    AVG_AGE
- ---------- ----------
F          30          38
M         186          43
```

Example 5-3 Using Some Predictors and an Expression

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
     FROM mining_data_apply_v
     WHERE PREDICTION(dt_sh_clas_sample USING
                     cust_gender, cust_marital_status, occupation,
                    'L: 300,000 and above' AS cust_income_level) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

The output is follows:

```
C         CNT    AVG_AGE
- ---------- ----------
F          30          38
M         186          43
```

5.2.2 Single-Record Scoring

You can score a single record which produces 0 and 1 to predict customers who are unlikely or likely to use an affinity card.

The Oracle Machine Learning for SQL functions can produce a score for a single record, as shown in Example 5-4 and Example 5-5.

Example 5-4 returns a prediction for customer 102001 by applying the classification model NB_SH_Clas_sample. The resulting score is 0, meaning that this customer is unlikely to use an affinity card. The NB_SH_Clas_Sample model is created by the `oml4sql-classification-naive-bayes.sql` example.

Example 5-5 returns a prediction for 'Affinity card is great' as the comments attribute by applying the text machine learning model T_SVM_Clas_sample. The resulting score is 1, meaning that this customer is likely to use an affinity card. The T_SVM_Clas_sample model is created by the `oml4sql-classification-text-analysis-svm.sql` example.

Example 5-4 Scoring a Single Customer or a Single Text Expression

```
SELECT PREDICTION (NB_SH_Clas_Sample USING *)
    FROM sh.customers where cust_id = 102001;
```

The output is as follows:

```
PREDICTION(NB_SH_CLAS_SAMPLEUSING*)
-----------------------------------
                                  0
```

Example 5-5 Scoring a Single Text Expression

```
SELECT
  PREDICTION(T_SVM_Clas_sample USING 'Affinity card is great' AS comments)
FROM DUAL;
```

The output is as follows:

```
PREDICTION(T_SVM_CLAS_SAMPLEUSING'AFFINITYCARDISGREAT'ASCOMMENTS)
-----------------------------------------------------------------
                                                                1
```

5.3 Prediction Details

Prediction details are XML strings that provide information about the score.

Details are available for all types of scoring: clustering, feature extraction, classification, regression, and anomaly detection. Details are available whether scoring is dynamic or the result of model apply.

The details functions, `CLUSTER_DETAILS`, `FEATURE_DETAILS`, and `PREDICTION_DETAILS` return the actual value of attributes used for scoring and the relative importance of the attributes in determining the score. By default, the functions return the five most important attributes in descending order of importance.

- Cluster Details
  Shows an example of the `CLUSTER_DETAILS` function.
- Feature Details
  Shows an example of the `FEATURE_DETAILS` function.
- Prediction Details
  Shows an examples of `PREDICTION_DETAILS` function.
- GROUPING Hint
  OML4SQL functions include `PREDICTION*`, `CLUSTER*`, `FEATURE*`, and `ORA_DM_*`. The `GROUPING` hint is an optional hint that applies to machine learning scoring functions when scoring partitioned models.

5.3.1 Cluster Details

Shows an example of the `CLUSTER_DETAILS` function.

For the most likely cluster assignments of customer 100955 (probability of assignment > 20%), the query in the following example produces the five attributes that have the most impact for each of the likely clusters. The clustering functions apply an Expectation Maximization model named em_sh_clus_sample to the data selected from `mining_data_apply_v`. The "5" specified in `CLUSTER_DETAILS` is not required, because five attributes are returned by default. The em_sh_clus_sample model is created by the `oml4sql-clustering-expectation-maximization.sql` example.

Example 5-6 Cluster Details

```
SELECT S.cluster_id, probability prob,
          CLUSTER_DETAILS(em_sh_clus_sample, S.cluster_id, 5 USING T.*) det
    FROM
      (SELECT v.*, CLUSTER_SET(em_sh_clus_sample, NULL, 0.2 USING *) pset
        FROM mining_data_apply_v v
       WHERE cust_id = 100955) T,
      TABLE(T.pset) S
    ORDER BY 2 DESC;
```

The output is as follows:

```
CLUSTER_ID  PROB DET
---------- -----
--------------------------------------------------------------------------------
        14 .6761 <Details algorithm="Expectation Maximization" cluster="14">
                 <Attribute name="AGE" actualValue="51" weight=".676"
rank="1"/>
                 <Attribute name="HOME_THEATER_PACKAGE" actualValue="1"
weight=".557" rank="2"/>
                 <Attribute name="FLAT_PANEL_MONITOR" actualValue="0"
weight=".412" rank="3"/>
                 <Attribute name="Y_BOX_GAMES" actualValue="0" weight=".171"
rank="4"/>
                 <Attribute name="BOOKKEEPING_APPLICATION"actualValue="1"
weight="-.003"
                  rank="5"/>
                 </Details>

         3 .3227 <Details algorithm="Expectation Maximization" cluster="3">
                 <Attribute name="YRS_RESIDENCE" actualValue="3"
weight=".323" rank="1"/>
                 <Attribute name="BULK_PACK_DISKETTES" actualValue="1"
weight=".265" rank="2"/>
                 <Attribute name="EDUCATION" actualValue="HS-grad"
weight=".172" rank="3"/>
                 <Attribute name="AFFINITY_CARD" actualValue="0"
weight=".125" rank="4"/>
                 <Attribute name="OCCUPATION" actualValue="Crafts"
weight=".055" rank="5"/>
                 </Details>
```

5.3.2 Feature Details

Shows an example of the FEATURE_DETAILS function.

The query in the following example returns the three attributes that have the greatest impact on the top Principal Components Analysis (PCA) projection for customer 101501. The FEATURE_DETAILS function applies a Singular Value Decomposition (SVD) model named svd_sh_sample to the data selected from the svd_sh_sample_build_num table. The table and model are created by the oml4sql-singular-value-decomposition.sql example.

Example 5-7 Feature Details

```
SELECT FEATURE_DETAILS(svd_sh_sample, 1, 3 USING *) proj1det
  FROM svd_sh_sample_build_num
  WHERE CUST_ID = 101501;
```

The output is as follows:

```
PROJ1DET
--------------------------------------------------------------------------------
--
<Details algorithm="Singular Value Decomposition" feature="1">
<Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".352"
rank="1"/>
<Attribute name="Y_BOX_GAMES" actualValue="0" weight=".249" rank="2"/>
<Attribute name="AGE" actualValue="41" weight=".063" rank="3"/>
</Details>
```

5.3.3 Prediction Details

Shows an examples of PREDICTION_DETAILS function.

The query in the following example returns the attributes that are most important in predicting the age of customer 100010. The prediction functions apply a Generalized Linear Model regression model named GLMR_SH_Regr_sample to the data selected from mining_data_apply_v. The GLMR_SH_Regr_sample model is created by the oml4sql-regression-glm.sql example.

Example 5-8 Prediction Details for Regression

```
SELECT cust_id,
      PREDICTION(GLMR_SH_Regr_sample USING *) pr,
      PREDICTION_DETAILS(GLMR_SH_Regr_sample USING *) pd
  FROM mining_data_apply_v
  WHERE CUST_ID = 100010;
```

The output is as follows:

```
CUST_ID    PR PD
------- ----- -----------
 100010 25.45 <Details algorithm="Generalized Linear Model">
              <Attribute name="FLAT_PANEL_MONITOR" actualValue="1"
weight=".025" rank="1"/>
              <Attribute name="OCCUPATION" actualValue="Crafts" weight=".019"
```

```
rank="2"/>
                <Attribute name="AFFINITY_CARD" actualValue="0" weight=".01"
rank="3"/>
                <Attribute name="OS_DOC_SET_KANJI" actualValue="0" weight="0"
rank="4"/>
                <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight="-.004" rank="5"/>
                </Details>
```

The query in the following example returns the customers who work in Tech Support and are likely to use an affinity card (with more than 85% probability). The prediction functions apply an Support Vector Machine (SVM) classification model named svmc_sh_clas_sample. to the data selected from `mining_data_apply_v`. The query includes the prediction details, which show that education is the most important predictor. The svmc_sh_clas_sample model is created by the `oml4sql-classification-svm.sql` example.

Example 5-9 Prediction Details for Classification

```
SELECT cust_id, PREDICTION_DETAILS(svmc_sh_clas_sample, 1 USING *) PD
      FROM mining_data_apply_v
  WHERE PREDICTION_PROBABILITY(svmc_sh_clas_sample, 1 USING *) > 0.85
  AND occupation = 'TechSup'
  ORDER BY cust_id;
```

The output is as follows:

```
CUST_ID PD
-------
--------------------------------------------------------------------------------
---------
 100029 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A" weight=".199"
rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="I: 170\,000 -
189\,999" weight=".044"
         rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".028"
rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".024"
rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".022" rank="5"/>
        </Details>

 100378 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A" weight=".21"
rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="B: 30\,000 -
49\,999" weight=".047"
         rank="2"/>
        <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".043"
rank="3"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".03"
rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
```

```
weight=".023" rank="5"/>
        </Details>

 100508 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Bach." weight=".19"
rank="1"/>
        <Attribute name="CUST_INCOME_LEVEL" actualValue="L: 300\,000 and
above" weight=".046"
         rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".031"
rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".026"
rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".024" rank="5"/>
        </Details>

 100980 <Details algorithm="Support Vector Machines" class="1">
        <Attribute name="EDUCATION" actualValue="Assoc-A" weight=".19"
rank="1"/>
        <Attribute name="FLAT_PANEL_MONITOR" actualValue="0" weight=".038"
rank="2"/>
        <Attribute name="HOME_THEATER_PACKAGE" actualValue="1" weight=".026"
rank="3"/>
        <Attribute name="BULK_PACK_DISKETTES" actualValue="1" weight=".022"
rank="4"/>
        <Attribute name="BOOKKEEPING_APPLICATION" actualValue="1"
weight=".02" rank="5"/>
        </Details>
```

The query in the following example returns the two customers that differ the most from the rest of the customers. The prediction functions apply an anomaly detection model named SVMO_SH_Clas_sample to the data selected from `mining_data_apply_v`. anomaly detection uses a one-class SVM classifier. The model is created by the `oml4sql-singular-value-decomposition.sql` example.

Example 5-10 Prediction Details for Anomaly Detection

```
SELECT cust_id, pd FROM
  (SELECT cust_id,
         PREDICTION_DETAILS(SVMO_SH_Clas_sample, 0 USING *) pd,
         RANK() OVER (ORDER BY prediction_probability(
               SVMO_SH_Clas_sample, 0 USING *) DESC, cust_id) rnk
  FROM mining_data_one_class_v)
  WHERE rnk <= 2
  ORDER BY rnk;
```

The output is as follows:

```
  CUST_ID PD
----------
--------------------------------------------------------------------------------
-----
    102366 <Details algorithm="Support Vector Machines" class="0">
           <Attribute name="COUNTRY_NAME" actualValue="United Kingdom"
weight=".078" rank="1"/>
```

```
           <Attribute name="CUST_MARITAL_STATUS" actualValue="Divorc."
weight=".027" rank="2"/>
           <Attribute name="CUST_GENDER" actualValue="F" weight=".01"
rank="3"/>
           <Attribute name="HOUSEHOLD_SIZE" actualValue="9+" weight=".009"
rank="4"/>
           <Attribute name="AGE" actualValue="28" weight=".006" rank="5"/>
           </Details>

    101790 <Details algorithm="Support Vector Machines" class="0">
           <Attribute name="COUNTRY_NAME" actualValue="Canada" weight=".068"
rank="1"/>
           <Attribute name="HOUSEHOLD_SIZE" actualValue="4-5" weight=".018"
rank="2"/>
           <Attribute name="EDUCATION" actualValue="7th-8th" weight=".015"
rank="3"/>
           <Attribute name="CUST_GENDER" actualValue="F" weight=".013"
rank="4"/>
           <Attribute name="AGE" actualValue="38" weight=".001" rank="5"/>
           </Details>
```

5.3.4 GROUPING Hint

OML4SQL functions include `PREDICTION*`, `CLUSTER*`, `FEATURE*`, and `ORA_DM_*`. The `GROUPING` hint is an optional hint that applies to machine learning scoring functions when scoring partitioned models.

This hint results in partitioning the input data set into distinct data slices so that each partition is scored in its entirety before advancing to the next partition. However, parallelism by partition is still available. Data slices are determined by the partitioning key columns used when the model was built. This method can be used with any machine learning function against a partitioned model. The hint may yield a query performance gain when scoring large data that is associated with many partitions but may negatively impact performance when scoring large data with few partitions on large systems. Typically, there is no performance gain if you use the hint for single row queries.

Enhanced PREDICTION Function Command Format

```
<prediction function> ::=
    PREDICTION <left paren> /*+ GROUPING */ <prediction model>
        [ <comma> <class value> [ <comma> <top N> ] ]
        USING <machine learning attribute list> <right paren>
```

The syntax for only the `PREDICTION` function is given but it is applicable to any machine learning function in which `PREDICTION`, `CLUSTERING`, and `FEATURE_EXTRACTION` scoring functions occur.

Example 5-11 Example

```
SELECT PREDICTION(/*+ GROUPING */my_model USING *) pred FROM <input table>;
```

Related Topics

- *Oracle Database SQL Language Reference*

5.4 Real-Time Scoring

You can perform real-time scoring by running a SQL query. An example shows a real-time query using `PREDICTION_PROBABILITY` function. Based on the result, a customer representative can offer a value card to the customer.

Oracle Machine Learning for SQL functions enable prediction, clustering, and feature extraction analysis to be easily integrated into live production and operational systems. Because machine learning results are returned within SQL queries, machine learning can occur in real time.

With real-time scoring, point-of-sales database transactions can be mined. Predictions and rule sets can be generated to help front-line workers make better analytical decisions. Real-time scoring enables fraud detection, identification of potential liabilities, and recognition of better marketing and selling opportunities.

The query in the following example uses a Decision Tree model named `dt_sh_clas_sample` to predict the probability that customer 101488 uses an affinity card. A customer representative can retrieve this information in real time when talking to this customer on the phone. Based on the query result, the representative can offer an extra-value card, since there is a 73% chance that the customer uses a card. The model is created by the `oml4sql-classification-decision-tree.sql` example.

Example 5-12 Real-Time Query with Prediction Probability

```
SELECT PREDICTION_PROBABILITY(dt_sh_clas_sample, 1 USING *) cust_card_prob
       FROM mining_data_apply_v
       WHERE cust_id = 101488;
```

The output is as follows:

```
CUST_CARD_PROB
--------------
        .72764
```

5.5 Dynamic Scoring

You can perform dynamic scoring if, for some reason, you do not want to apply a predefined model.

The Oracle Machine Learning for SQL functions operate in two modes: by applying a predefined model, or by executing an analytic clause. If you supply an analytic clause instead of a model name, the function builds one or more transient models and uses them to score the data.

The ability to score data dynamically without a predefined model extends the application of basic embedded machine learning techniques into environments where models are not available. Dynamic scoring, however, has limitations. The transient models created during dynamic scoring are not available for inspection or fine tuning. Applications that require model inspection, the correlation of scoring results with the model, special algorithm settings, or multiple scoring queries that use the same model, require a predefined model.

The following example shows a dynamic scoring query. The example identifies the rows in the input data that contain unusual customer age values.

Example 5-13 Dynamic Prediction

```
SELECT cust_id, age, pred_age, age-pred_age age_diff, pred_det FROM
 (SELECT cust_id, age, pred_age, pred_det,
    RANK() OVER (ORDER BY ABS(age-pred_age) DESC) rnk FROM
    (SELECT cust_id, age,
         PREDICTION(FOR age USING *) OVER () pred_age,
         PREDICTION_DETAILS(FOR age ABS USING *) OVER () pred_det
  FROM mining_data_apply_v))
WHERE rnk <= 5;
```

The output is follows:

```
CUST_ID  AGE   PRED_AGE AGE_DIFF PRED_DET
------- ---- ---------- --------
-------------------------------------------------------------
 100910   80 40.6686505    39.33 <Details algorithm="Support Vector Machines">
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="Y_BOX_GAMES" actualValue="0"
                                  weight=".059" rank="2"/>
                                 <Attribute name="AFFINITY_CARD"
actualValue="0"
                                  weight=".059" rank="3"/>
                                 <Attribute name="FLAT_PANEL_MONITOR"
actualValue="1"
                                  weight=".059" rank="4"/>
                                 <Attribute name="YRS_RESIDENCE"
actualValue="4"
                                  weight=".059" rank="5"/>
                                  </Details>

 101285   79 42.1753571    36.82 <Details algorithm="Support Vector Machines">
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="HOUSEHOLD_SIZE"
actualValue="2" weight=".059"
                                  rank="2"/>
                                 <Attribute name="CUST_MARITAL_STATUS"
actualValue="Mabsent"
                                  weight=".059" rank="3"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="4"/>
                                 <Attribute name="OCCUPATION"
actualValue="Prof." weight=".059"
                                  rank="5"/>
                                 </Details>

 100694   77 41.0396722    35.96 <Details algorithm="Support Vector Machines">
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="EDUCATION"
```

```
actualValue="&lt; Bach."
                                  weight=".059" rank="2"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="3"/>
                                 <Attribute name="CUST_ID"
actualValue="100694" weight=".059"
                                  rank="4"/>
                                 <Attribute name="COUNTRY_NAME"
actualValue="United States of
                                  America" weight=".059" rank="5"/>
                                 </Details>

 100308   81 45.3252491    35.67 <Details algorithm="Support Vector Machines">
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="1"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="2"/>
                                 <Attribute name="HOUSEHOLD_SIZE"
actualValue="2" weight=".059"
                                  rank="3"/>
                                 <Attribute name="FLAT_PANEL_MONITOR"
actualValue="1"
                                  weight=".059" rank="4"/>
                                 <Attribute name="CUST_GENDER"
actualValue="F" weight=".059"
                                  rank="5"/>
                                 </Details>

 101256   90 54.3862214    35.61 <Details algorithm="Support Vector Machines">
                                 <Attribute name="YRS_RESIDENCE"
actualValue="9" weight=".059"
                                  rank="1"/>
                                 <Attribute name="HOME_THEATER_PACKAGE"
actualValue="1"
                                  weight=".059" rank="2"/>
                                 <Attribute name="EDUCATION"
actualValue="&lt; Bach."
                                  weight=".059" rank="3"/>
                                 <Attribute name="Y_BOX_GAMES"
actualValue="0" weight=".059"
                                  rank="4"/>
                                 <Attribute name="COUNTRY_NAME"
actualValue="United States of
                                  America" weight=".059" rank="5"/>
                                 </Details>
```

5.6 Cost-Sensitive Decision Making

Costs are user-specified numbers that bias classification. The algorithm uses positive numbers to penalize more expensive outcomes over less expensive outcomes. Higher numbers indicate higher costs.

The algorithm uses negative numbers to favor more beneficial outcomes over less beneficial outcomes. Lower negative numbers indicate higher benefits.

All classification algorithms can use costs for scoring. You can specify the costs in a cost matrix table, or you can specify the costs inline when scoring. If you specify costs inline and the model also has an associated cost matrix, only the inline costs are used. The `PREDICTION`, `PREDICTION_SET`, and `PREDICTION_COST` functions support costs.

Only the Decision Tree algorithm can use costs to bias the model build. If you want to create a Decision Tree model with costs, create a cost matrix table and provide its name in the `CLAS_COST_TABLE_NAME` setting for the model. If you specify costs when building the model, the cost matrix used to create the model is used when scoring. If you want to use a different cost matrix table for scoring, first remove the existing cost matrix table then add the new one.

A sample cost matrix table is shown in the following table. The cost matrix specifies costs for a binary target. The matrix indicates that the algorithm must treat a misclassified 0 as twice as costly as a misclassified 1.

Table 5-1 Sample Cost Matrix

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	COST
0	0	0
0	1	2
1	0	1
1	1	0

Example 5-14 Sample Queries With Costs

The table `nbmodel_costs` contains the cost matrix described in Table 5-1.

```
SELECT * from nbmodel_costs;
```

The output is as follows:

```
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1          2
                  1                      0          1
                  1                      1          0
```

The following statement associates the cost matrix with a Naive Bayes model called `nbmodel`.

```
BEGIN
  dbms_data_mining.add_cost_matrix('nbmodel', 'nbmodel_costs');
END;
/
```

The following query takes the cost matrix into account when scoring `mining_data_apply_v`. The output is restricted to those rows where a prediction of 1 is less costly then a prediction of 0.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
        FROM mining_data_apply_v
        WHERE PREDICTION (nbmodel COST MODEL
       USING cust_marital_status, education, household_size) = 1
```

```
        GROUP BY cust_gender
        ORDER BY cust_gender;
```

The output is as follows:

```
C         CNT    AVG_AGE
- ---------- ----------
F          25         38
M         208         43
```

You can specify costs inline when you invoke the scoring function. If you specify costs inline and the model also has an associated cost matrix, only the inline costs are used. The same query is shown below with different costs specified inline. Instead of the "2" shown in the cost matrix table (Table 5-1), "10" is specified in the inline costs.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
      FROM mining_data_apply_v
      WHERE PREDICTION (nbmodel
                COST (0,1) values ((0, 10),
                           (1, 0))
                USING cust_marital_status, education, household_size) = 1
      GROUP BY cust_gender
      ORDER BY cust_gender;
```

The output is as follows:

```
C         CNT    AVG_AGE
- ---------- ----------
F          74         39
M         581         43
```

The same query based on probability instead of costs is shown below.

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

The output is as follows:

```
C         CNT    AVG_AGE
- ---------- ----------
F          73         39
M         577         44
```

Related Topics

- Example 1-1

5.7 DBMS_DATA_MINING.APPLY

The APPLY procedure in DBMS_DATA_MINING is a batch apply operation that writes the results of scoring directly to a table.

The columns in the table are machine learning function-dependent.

Scoring with APPLY generates the same results as scoring with the SQL scoring functions. Classification produces a prediction and a probability for each case; clustering produces a cluster ID and a probability for each case, and so on. The difference lies in the way that scoring results are captured and the mechanisms that can be used for retrieving them.

APPLY creates an output table with the columns shown in the following table:

Table 5-2 APPLY Output Table

Machine Learning Technique	Output Columns
classification	CASE_ID PREDICTION PROBABILITY
regression	CASE_ID PREDICTION
anomaly detection	CASE_ID PREDICTION PROBABILITY
clustering	CASE_ID CLUSTER_ID PROBABILITY
feature extraction	CASE_ID FEATURE_ID MATCH_QUALITY

Since APPLY output is stored separately from the scoring data, it must be joined to the scoring data to support queries that include the scored rows. Thus any model that is used with APPLY must have a case ID.

A case ID is not required for models that is applied with SQL scoring functions. Likewise, storage and joins are not required, since scoring results are generated and consumed in real time within a SQL query.

The following example illustrates anomaly detection with APPLY. The query of the APPLY output table returns the ten first customers in the table. Each has a a probability for being typical (1) and a probability for being anomalous (0). The SVMO_SH_Clas_sample model is created by the oml4sql-anomaly-detection-1class-svm.sql example.

Example 5-15 Anomaly Detection with DBMS_DATA_MINING.APPLY

```
EXEC dbms_data_mining.apply
        ('SVMO_SH_Clas_sample','svmo_sh_sample_prepared',
         'cust_id', 'one_class_output');

SELECT * from one_class_output where rownum < 11;
```

The output is as follows:

```
   CUST_ID PREDICTION PROBABILITY
---------- ---------- -----------
    101798          1  .567389309
    101798          0  .432610691
    102276          1  .564922469
    102276          0  .435077531
    102404          1   .51213544
    102404          0   .48786456
    101891          1  .563474346
    101891          0  .436525654
    102815          0  .500663683
    102815          1  .499336317
```

Related Topics

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

6

Machine Learning Operations on Unstructured Text

Explains how to use Oracle Machine Learning for SQL to operate on unstructured text.

- About Unstructured Text
  Unstructured text may contain important information that is critical to the success of a business.
- About Machine Learning and Oracle Text
  Understand machine learning operations on text and Oracle Text.
- Create a Model that Includes Machine Learning Operations on Text
  Create a model and specify the settings to perform machine learning operations on text.
- Create a Text Policy
  An Oracle Text policy specifies how text content must be interpreted. You can provide a text policy to govern a model, an attribute, or both the model and individual attributes.
- Configure a Text Attribute
  Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.

6.1 About Unstructured Text

Unstructured text may contain important information that is critical to the success of a business.

Machine learning algorithms act on data that is numerical or categorical. Numerical data is ordered. It is stored in columns that have a numeric data type, such as `NUMBER` or `FLOAT`. Categorical data is identified by category or classification. It is stored in columns that have a character data type, such as `VARCHAR2` or `CHAR`.

Unstructured text data is neither numerical nor categorical. Unstructured text includes items such as web pages, document libraries, Power Point presentations, product specifications, emails, comment fields in reports, and call center notes. It has been said that unstructured text accounts for more than three quarters of all enterprise data. Extracting meaningful information from unstructured text can be critical to the success of a business.

6.2 About Machine Learning and Oracle Text

Understand machine learning operations on text and Oracle Text.

Machine learning operations on text is the process of applying machine learning techniques to text terms, also called text features or tokens. Text terms are words or groups of words that have been extracted from text documents and assigned numeric weights. Text terms are the fundamental unit of text that can be manipulated and analyzed.

Oracle Text is an Oracle AI Database technology that provides term extraction, word and theme searching, and other utilities for querying text. When columns of text are present in the training data, Oracle Machine Learning for SQL uses Oracle Text utilities and term weighting strategies to transform the text for machine learning operations. OML4SQL passes

configuration information supplied by you to Oracle Text and uses the results in the model creation process.

Related Topics

- *Oracle Text Application Developer's Guide*

6.3 Create a Model that Includes Machine Learning Operations on Text

Create a model and specify the settings to perform machine learning operations on text.

Oracle Machine Learning for SQL supports unstructured text within columns of `VARCHAR2`, `CHAR`, `CLOB`, `BLOB`, and `BFILE`, as described in the following table:

Table 6-1 Column Data Types That May Contain Unstructured Text

Data Type	Description
`BFILE` and `BLOB`	Oracle Machine Learning for SQL interprets `BLOB` and `BFILE` as text *only if* you identify the columns as text when you create the model. If you do not identify the columns as text, then `CREATE_MODEL` returns an error.
`CLOB`	OML4SQL interprets `CLOB` as text.
`CHAR`	OML4SQL interprets `CHAR` as categorical by default. You can identify columns of `CHAR` as text when you create the model.
`VARCHAR2`	OML4SQL interprets `VARCHAR2` with data length > 4000 as text. OML4SQL interprets `VARCHAR2` with data length <= 4000 as categorical by default. You can identify these columns as text when you create the model.

> **Note**
>
> Text is not supported in nested columns or as a target in supervised machine learning.

The settings described in the following table control the term extraction process for text attributes in a model. Instructions for specifying model settings are in "Specifying Model Settings".

Table 6-2 Model Settings for Text

Setting Name	Data Type	Setting Value	Description
`ODMS_TEXT_POLICY_NAME`	`VARCHAR2(4000)`	Name of an Oracle Text policy object created with `CTX_DDL.CREATE_POLICY`	Affects how individual tokens are extracted from unstructured text.
`ODMS_TEXT_MAX_FEATURES`	`INTEGER`	1 <= *value* <= 100000	Maximum number of features to use from the document set (across all documents of each text column) passed to `CREATE_MODEL`. Default is 3000.

A model can include one or more text attributes. A model with text attributes can also include categorical and numerical attributes.

To create a model that includes text attributes:

1. Create an Oracle Text policy object.
2. Specify the model configuration settings that are described in "Table 6-2".
3. Specify which columns must be treated as text and, optionally, provide text transformation instructions for individual attributes.
4. Pass the model settings and text transformation instructions to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

> **Note**
>
> All algorithms except O-Cluster can support columns of unstructured text.
>
> The use of unstructured text is not recommended for association rules (Apriori).

In the following example, an SVM model is used to predict customers that are most likely to be positive responders to an Affinity Card loyalty program. The data comes with a text column that contains user generated comments. By creating an Oracle Text policy and specifying model settings, the algorithm automatically uses the text column and builds the model on both the structured data and unstructured text.

This example uses a view called `mining_data` which is created from `SH.SALES` table. A training data set called `mining_train_text` is also created.

The following queries show you how to create an Oracle Text policy followed by building a model using `CREATE_MODEL2` procedure.

```
%script

BEGIN

EXECUTE ctx_ddl.create_policy('dmdemo_svm_policy');
```

The output is:

```
PL/SQL procedure successfully completed.

----------------------------

PL/SQL procedure successfully completed.
```

```
%script

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

BEGIN

    v_setlst(dbms_data_mining.algo_name) :=
dbms_data_mining.algo_support_vector_machines;
```

```
    v_setlst(dbms_data_mining.prep_auto) :=  dbms_data_mining.prep_auto_on;
    v_setlst(dbms_data_mining.svms_kernel_function) := dbms_data_mining.svms_linear;
    v_setlst(dbms_data_mining.svms_complexity_factor) := '100';
    v_setlst(dbms_data_mining.odms_text_policy_name) := 'DMDEMO_SVM_POLICY';

    v_setlst(dbms_data_mining.svms_solver) :=  dbms_data_mining.svms_solver_sgd;
    dbms_data_mining_transform.SET_TRANSFORM(
        xformlist, 'comments', null, 'comments', null, 'TEXT');
    DBMS_DATA_MINING.CREATE_MODEL2(
        model_name          => 'T_SVM_Clas_sample',
        mining_function     => dbms_data_mining.classification,
        data_query          => 'select * from mining_train_text',
        set_list            => v_setlst,
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        xform_list => xformlist);
END;
/
```

The output is:

```
PL/SQL procedure successfully completed.

----------------------------

PL/SQL procedure successfully completed.

----------------------------
```

Related Topics

- Model Detail Views for Text Features
  The model details view for text features is `DM$VX`*model_name*.
- Specify Model Settings
  You can configure your model by specifying model settings.
- Create a Text Policy
  An Oracle Text policy specifies how text content must be interpreted. You can provide a text policy to govern a model, an attribute, or both the model and individual attributes.
- Configure a Text Attribute
  Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.
- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.

6.4 Create a Text Policy

An Oracle Text policy specifies how text content must be interpreted. You can provide a text policy to govern a model, an attribute, or both the model and individual attributes.

If a model-specific policy is present and one or more attributes have their own policies, Oracle Machine Learning for SQL uses the attribute policies for the specified attributes and the model-specific policy for the other attributes.

The `CTX_DDL.CREATE_POLICY` procedure creates a text policy.

```
CTX_DDL.CREATE_POLICY(
          policy_name    IN VARCHAR2,
                          filter         IN VARCHAR2 DEFAULT NULL,
                          section_group  IN VARCHAR2 DEFAULT NULL,
                          lexer          IN VARCHAR2 DEFAULT NULL,
                          stoplist       IN VARCHAR2 DEFAULT NULL,
                          wordlist       IN VARCHAR2 DEFAULT NULL);
```

The parameters of `CTX_DDL.CREATE_POLICY` are described in the following table.

Table 6-3 CTX_DDL.CREATE_POLICY Procedure Parameters

Parameter Name	Description
`policy_name`	Name of the new policy object. Oracle Text policies and text indexes share the same namespace.
`filter`	Specifies how the documents must be converted to plain text for indexing. Examples are: `CHARSET_FILTER` for character sets and `NULL_FILTER` for plain text, HTML and XML. For `filter` values, see "Filter Types" in *Oracle Text Reference*.
`section_group`	Identifies sections within the documents. For example, `HTML_SECTION_GROUP` defines sections in HTML documents. For `section_group` values, see "Section Group Types" in *Oracle Text Reference*. Note: You can specify any section group that is supported by `CONTEXT` indexes.
`lexer`	Identifies the language that is being indexed. For example, `BASIC_LEXER` is the lexer for extracting terms from text in languages that use white space delimited words (such as English and most western European languages). For `lexer` values, see "Lexer Types" in *Oracle Text Reference*.
`stoplist`	Specifies words and themes to exclude from term extraction. For example, the word "the" is typically in the stoplist for English language documents. The system-supplied stoplist is used by default. See "Stoplists" in *Oracle Text Reference*.
`wordlist`	Specifies how stems and fuzzy queries must be expanded. A stem defines a root form of a word so that different grammatical forms have a single representation. A fuzzy query includes common misspellings in the representation of a word. See "`BASIC_WORDLIST`" in *Oracle Text Reference*.

Related Topics

- *Oracle Text Reference*

6.5 Configure a Text Attribute

Provide transformation instructions for text attribute or unstructured text by explicitly identifying the column datatypes.

As shown in Table 6-1, you can identify columns of `CHAR`, shorter `VARCHAR2` (<=4000), `BFILE`, and `BLOB` as text attributes. If `CHAR` and shorter `VARCHAR2` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` processes them as categorical attributes. If `BFILE` and `BLOB` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` returns an error.

To identify a column as a text attribute, supply the keyword `TEXT` in an **Attribute specification**. The attribute specification is a field (`attribute_spec`) in a transformation record (`transform_rec`). Transformation records are components of transformation lists (`xform_list`) that can be passed to `CREATE_MODEL`or `CREATE_MODEL2`.

> **Note**
>
> An attribute specification can also include information that is not related to text. Instructions for constructing an attribute specification are in "Embedding Transformations in a Model".

You can provide transformation instructions for any text attribute by qualifying the `TEXT` keyword in the attribute specification with the subsettings described in the following table.

Table 6-4 Attribute-Specific Text Transformation Instructions

Subsetting Name	Description	Example
`BIGRAM`	A sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. Here, `NORMAL` tokens are mixed with their bigrams.	(`TOKEN_TYPE:`*`BIGRAM`*)
`POLICY_NAME`	Name of an Oracle Text policy object created with `CTX_DDL.CREATE_POLICY`	(`POLICY_NAME:`*`my_policy`*)
`STEM_BIGRAM`	Here, `STEM` tokens are extracted first and then stem bigrams are formed.	(`TOKEN_TYPE:`*`STEM_BIGRAM`*)
`SYNONYM`	Oracle Machine Learning for SQL supports synonyms. The following is an optional parameter: `<thesaurus>` where `<thesaurus>` is the name of the thesaurus defining synonyms. If `SYNONYM` is used without this parameter, then the default thesaurus is used.	(`TOKEN_TYPE:`*`SYNONYM`*) (`TOKEN_TYPE:`*`SYNONYM[NAMES]`*)

`TOKEN_TYPE`	The following values are supported:
	`NORMAL` (the default)
	`STEM`
	`THEME`
	See "Token Types in an Attribute Specification"
		(`TOKEN_TYPE:THEME`)

`MAX_FEATURES`	Maximum number of features to use from the attribute.	(`MAX_FEATURES:3000`)

> **Note**
>
> The `TEXT` keyword is only required for `CLOB` and longer `VARCHAR2` (>4000) when you specify transformation instructions. The `TEXT` keyword is *always* required for `CHAR`, shorter `VARCHAR2`, `BFILE`, and `BLOB` — whether or not you specify transformation instructions.

> **Tip**
>
> You can view attribute specifications in the data dictionary view `ALL_MINING_MODEL_ATTRIBUTES`, as shown in *Oracle Database Reference*.

Token Types in an Attribute Specification

When stems or themes are specified as the token type, the lexer preference for the text policy must support these types of tokens.

The following example adds themes and English stems to `BASIC_LEXER`.

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'BASIC_LEXER');
  CTX_DDL.SET_ATTRIBUTE('my_lexer', 'index_stems', 'ENGLISH');
  CTX_DDL.SET_ATTRIBUTE('my_lexer', 'index_themes', 'YES');
END;
```

Example 6-1 A Sample Attribute Specification for Text

This expression specifies that text transformation for the attribute must use the text policy named `my_policy`. The token type is `THEME`, and the maximum number of features is 3000.

```
"TEXT(POLICY_NAME:my_policy)(TOKEN_TYPE:THEME)(MAX_FEATURES:3000)"
```

Related Topics

- Embed Transformations in a Model
  You can specify your own transformations and embed them in a model by creating a transformation list and passing it to `DBMS_DATA_MINING.CREATE_MODEL2` or `DBMS_DATA_MINING.CREATE_MODEL`.
- Specify Transformation Instructions for an Attribute
  You can pass transformation instructions for an attribute by defining a transformation list.
- *Oracle Database PL/SQL Packages and Types Reference*
- ALL_MINING_MODEL_ATTRIBUTES

7

Integration of ONNX Runtime

Learn about ONNX Runtime that enables you to use ONNX models for machine learning tasks within your Oracle AI Database instance.

- About ONNX
  ONNX is an open-source format designed for machine learning models. It ensures cross-platform compatibility. This format also supports major languages and frameworks, facilitating efficient model exchange.
- Supported Machine Learning Functions for ONNX Runtime
  Describes the supported machine learning functions to import pretrained models and perform scoring.
- Supported Attribute Data Types
  Discover the supported ONNX input data types mapped to SQL data types.
- Supported Target Data Types
  Discover the supported ONNX target data types mapped to SQL data types.
- Custom ONNX Runtime Operations
  If you are looking to customize a pretrained embedding model by augmenting with pre-processing and post-processing operations, Oracle supports tokenization of an embedding model as a pre-processing operation and pooling and normalization as post-processing custom ONNX Runtime operations for version 1.15.1.
- Use PL/SQL Packages to Import Models
  Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure or the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to import ONNX format models. You can then use the imported ONNX format models through a scoring function run by the in-database ONNX Runtime.
- Supported SQL Scoring Functions
  Supported scoring functions for in-database scoring of machine learning models imported in the ONNX format are listed.
- Examples of Using ONNX Models
  The following examples use the Iris data set to showcase loading and inference from ONNX format machine learning models for machine learning techniques such as Classification, Regression, and Clustering in your Oracle AI Database instance.
- Traditional Machine Learning ONNX Format Models
  Traditional machine learning models using algorithms such as decision trees, random forests, and support vector machines, among others, can be converted to ONNX format. Such models may be produced in other environments and deployed through Oracle AI Database.
- Text Transformer ONNX Format Models
  Text transformers have the ability to translate natural language text into a numerical vector representation also known as an embedding, you use such vectors for semantic similarity search or other Natural Language Processing (NLP) use cases.
- Image Transformer ONNX Format Models
  Image transformer is a part of machine learning that helps computers interpret and analyze images and videos. It provides tools to perform tasks like creating image embeddings

(using an image transformer), classifying objects, detecting anomalies, and identifying objects in pictures or videos.

7.1 About ONNX

ONNX is an open-source format designed for machine learning models. It ensures cross-platform compatibility. This format also supports major languages and frameworks, facilitating efficient model exchange.

The ONNX format allows for model serialization. Models are represented as graphs of common machine learning operations. These graphs are saved in a portable format called protocol buffers. It simplifies the exchange of models across various platforms. These platforms include cloud, web, edge, and mobile experiences on Microsoft Windows, Linux, Mac, iOS, and Android. ONNX models also offer flexibility to export and import model in many languages such as Python, C++, C#, and Java to name a few. The ONNX format is useful for compute-heavy tasks such as training machine learning models and data processing that often uses trained models. Many leading machine learning development frameworks such as TensorFlow, Pytorch, and Scikit-learn, offer the capability to convert models into the ONNX format.

Once you represent the models in the ONNX format, you can run them with the ONNX Runtime. The architecture of the ONNX Runtime is adaptable, enabling providers to modify or enhance how some operations are implemented to make better use of particular hardware, such as, Graphical Processing Units (GPUs), Single Instruction Multiple Data (SIMD) instruction sets or specialized libraries. To learn more on ONNX Runtime, see https://onnxruntime.ai/docs/.

The ONNX Runtime integration with Oracle Database allows for the import of ONNX-formatted models, including embedding models. To support embedding models, Oracle Machine Learning has introduced a machine learning technique called *embedding*. You can only use ONNX embedding models in the Oracle database if they were converted using Oracle's Python utility package. Oracle's Python utility downloads a pretrained model, converts the model to ONNX format augmented with pre-processsing and post-processing operations and imports the ONNX format model to Oracle Database. For more information on the Python utility tool, see Convert Pretrained Models to ONNX Format.

Oracle supports ONNX Runtime version 1.20.1.

- Initializers
  Initializers are named constants stored inside ONNX models that are required during inference.
- External Initializers and In-Memory Sharing
  A model that uses very large initializers can reach the 2 GB size limit. To solve this problem, ONNX can store initializers as external files referred as external initializers or external data.
- Enable and Use External Initializers
  Use shared external initializers to reduce memory usage by loading large model data once into global memory. You verify eligibility, enable sharing, monitor usage, and later disable it when no longer needed.
- Generate External Initializers with OML4Py
  You can generate ONNX models with external initializers and their metadata using OML4Py.

7.1.1 Initializers

Initializers are named constants stored inside ONNX models that are required during inference.

Most commonly, these are the model parameters such as weights and biases of layers; however, they can also store other fixed constants such as normalization coefficients. Machine learning operations, such as matrix multiplication, use these initializers while running operations.

7.1.2 External Initializers and In-Memory Sharing

A model that uses very large initializers can reach the 2 GB size limit. To solve this problem, ONNX can store initializers as external files referred as external initializers or external data.

Advanced models may use large constant tensor values (multi-dimensional arrays) whose cumulated size may be larger than 2GB and prevents serializing these models into protocol buffers.

Embedding models, crucial for tasks like vector similarity search, have initializers that often account for 95% of their file size and may range from dozens of megabytes to several gigabytes. Until this release, the approach was to load a private copy of these initializers into each session's process memory (PGA), leading to high total memory usage for concurrent workloads.

In-Memory Sharing

Oracle's ONNX integration supports the sharing and in-memory population of external initializers, enabling the same model’s initializers to be loaded once into global memory and accessed by all database sessions needing that model. This improves memory efficiency and scalability.

Only ONNX models imported with external initializers are eligible for in-memory population. When enabled, the model's initializers are loaded into a shared global area for use by all qualifying processes.

See Enable and Use External Initializers and on how to enable and use shared external initilizers.

7.1.3 Enable and Use External Initializers

Use shared external initializers to reduce memory usage by loading large model data once into global memory. You verify eligibility, enable sharing, monitor usage, and later disable it when no longer needed.

Before You Begin

Models must use external initializers to qualify for sharing. You can verify the model compatibility and proceed with the next steps. Administrators and data scientists must ensure their model import or conversion pipelines are designed accordingly, leveraging Oracle's OML4Py utilities or their own frameworks to export ONNX models with external initializers and associated metadata. See Support For Large ONNX Format Model Support for details on how to create models with external data.

Follow the steps to verify, enable, and use external initializers:

1. Check that the ONNX model uses external initializers:

   ```
   SELECT model_name, EXTERNAL_DATA FROM all_mining_models;
   ```

   Only models with `EXTERNAL_DATA=YES` can be enabled for in-memory sharing.

2. Enable in-memory sharing.

   ```
   EXECUTE DBMS_DATA_MINING.INMEMORY_ONNX_MODEL('your_model_name');
   ```

   This loads the initializers into shared memory. The `INMEMORY` status on the model view is set to `YES`.

3. Monitor the model usage and memory.

   ```
   SELECT * FROM V$IM_ONNX_MODEL WHERE NAME = 'your_model_name';
   ```

   Shows metadata, population status, initializer sizes, and pin count. Inference in a session increments `PIN_COUNT`, displaying active usage of shared memory or many processes are sharing the model. If the session stops using the model for a while, the system decreases the pin count by one.

4. Use `V$IM_ONNX_SEGMENT` for deeper insights for troubleshooting.

   ```
   SELECT * FROM V$IM_ONNX_SEGMENT WHERE NAME = 'your_model_name';
   ```

5. Disable and release the memory.

   ```
   EXECUTE DBMS_DATA_MINING.INMEMORY_ONNX_MODEL('your_model_name', enable =>
   FALSE);
   ```

> **Note**
>
> The system releases the memory only after all sessions remove their pin.

> **See Also**
>
> - INMEMORY_ONNX_MODEL Procedure to learn more about enabling in-memory sharing of ONNX models with external initializers.
> - ALL_MINING_MODELS to learn about the dictionary views for parameter related to in-memory sharing.
> - V_IM_ONNX_MODEL to learn about the dynamic views related to in-memory sharing.
> - V_IM_ONNX_SEGMENT to learn about the dynamic views related to in-memory sharing.

7.1.4 Generate External Initializers with OML4Py

You can generate ONNX models with external initializers and their metadata using OML4Py.

You can import those ONNX models into your database. OML4Py also creates a metadata file that describes each initializer, making it easy for the ONNX runtime or Oracle AI Database to work with your model. See Support For Large ONNX Format Model Support on using OML4Py external initializers.

OML4Py enables importing or exporting models with external initializers using the following structure:

- A `.onnx` file that holds the ONNX model referencing external data.
- One or more `.data` files containing raw tensor data values.
- A single `.json` file describing the metadata of the external initializers. This JSON file includes the name, shape, offset, type, and size for each initializer.

See Also

- See IMPORT_ONNX_MODEL Procedure to import such models.
- View model details by querying `DM$VX<model_name>` and `DM$VG<model_name>`. See Model Detail Views for ONNX Models for more details.

7.2 Supported Machine Learning Functions for ONNX Runtime

Describes the supported machine learning functions to import pretrained models and perform scoring.

The following are the supported machine learning functions:

- Classification
- Clustering
- Embedding
- Regression

7.3 Supported Attribute Data Types

Discover the supported ONNX input data types mapped to SQL data types.

Data Type	SQL Type	Supported ONNX Data Type
Numerical	`BINARY_DOUBLE` `NUMBER`	`float`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`
Categorical	`VARCHAR`	For `VARCHAR` type: `string`
Text	`VARCHAR2` `CLOB`	`string`

Data Type	SQL Type	Supported ONNX Data Type
Vectors	`VECTOR(float32,<dimension>)`	`float`

The following data types are not supported:

- `complex64`, `complex128`
- `float16`, `bfloat16`
- `fp8`
- `int4`, `uint4`

7.4 Supported Target Data Types

Discover the supported ONNX target data types mapped to SQL data types.

Depending on the machine learning function, different scoring functions are used. Different scoring function for same machine learning function can produce different data types. A few points to note:

- Classification models have different rules to determine the type of `PREDICTION` function to be used. If you are using `PREDICTION_PROBABILITY`, then `BINARY_DOUBLE` is returned. See labels in JSON Metadata Parameters for ONNX Models.
- For an embedding model, the `VECTOR_EMBEDDING` function returns a `VECTOR` type.
- For a regression model, `VARCHAR` is not a valid target type and `BINARY_DOUBLE` is returned.
- For a clustering model, if you are using `CLUSTERING_PROBABILITY` and `CLUSTER_DISTANCE`, then `BINARY_DOUBLE` is returned.

To learn more, see JSON Metadata Parameters for ONNX Models

Machine Learning Function	SQL Function	SQL Type	Supported ONNX Target Output
Regression	`PREDICTION`	`BINARY_DOUBLE`	`regressionOutput`
Classification	`PREDICTION`	`VARCHAR2`	`classificationLabelOutput`
Classification	`PREDICTION`	`NUMBER`	`classificationLabelOutput`
Classification	`PREDICTION_PROBABILITY`	`BINARY_DOUBLE`	`classificationProbOutput`
Classification	`PREDICTION_SET`	`set of ( NUMBER , BINARY_DOUBLE ) set of (target_type, BINARY_DOUBLE)`	NA
Clustering	`CLUSTER_PROBABILITY`	`BINARY_DOUBLE`	`clusteringProbOutput`
Clustering	`CLUSTER_DISTANCE`	`BINARY_DOUBLE`	`clusteringDistanceOutput`

Machine Learning Function	SQL Function	SQL Type	Supported ONNX Target Output
Clustering	CLUSTER_SET	set of (NUMBER , BINARY_DOUBLE)	NA
Embedding	VECTOR_EMBEDDING	VECTOR(float32, n)	embeddingOutput

7.5 Custom ONNX Runtime Operations

If you are looking to customize a pretrained embedding model by augmenting with pre-processing and post-processing operations, Oracle supports tokenization of an embedding model as a pre-processing operation and pooling and normalization as post-processing custom ONNX Runtime operations for version 1.15.1.

Oracle offers a Python utility that provides a mechanism to augment a pretrained model with tokenization, pooling and normalization. The Python utility can augment the model with pre-processing and post-processing operations and convert a pretrained model to an ONNX format. Models using any other custom operations will fail on import. For details on how to use the Python utility, see Convert Pretrained Models to ONNX Format.

7.6 Use PL/SQL Packages to Import Models

Use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure or the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to import ONNX format models. You can then use the imported ONNX format models through a scoring function run by the in-database ONNX Runtime.

- To import a pretrained ONNX format model, use IMPORT_ONNX_MODEL Procedure or LOAD_ONNX_MODEL Procedure.
- To drop an ONNX model, use DROP_ONNX_MODEL. See also DROP_MODEL procedure.
- A complete step-by-step example that illustrates these procedures is in Import ONNX Models and Generate Embeddings.

> **Note**
>
> In-database embedding models must include tokenization and postprocessing. Providing only the core ONNX model is insufficient, as users would need to handle tokenization externally, pass tensors into the SQL operator, and convert output tensors into vectors.

The `DBMS_DATA_MINING.RENAME_MODEL` procedure is also supported.

Most of the existing Oracle Machine Learning for SQL APIs are available to the ONNX models. As partitioning is not applicable for external pretrained models, ONNX models do not support the following procedures:

- `ADD_PARTITION`
- `DROP_PARTITION`
- `ADD_COST_MATRIX`

- REMOVE_COST_MATRIX

Related Topics

- Summary of DBMS_DATA_MINING Subprograms

7.7 Supported SQL Scoring Functions

Supported scoring functions for in-database scoring of machine learning models imported in the ONNX format are listed.

Machine Learning Technique	Operator	Supported	Return Type
Embedding	VECTOR_EMBEDDING	always	VECTOR(<dimensions, FLOAT32>) The number of dimensions of the output vector of a VECTOR_EMBEDDING operator is defined by the embedding models.
Regression	PREDICTION	always	Data type of the target. For regression, the data type is converted to BINARY_DOUBLE SQL type.
Classification	PREDICTION	always	Data type of the target.
Classification	PREDICTION_PROBABILITY	always	BINARY_DOUBLE
Classification	PREDICTION_SET	always	Set of (t, NUMBER , BINARY_DOUBLE) where t is the data type of the target.
Clustering	CLUSTER_ID	only if clusteringProbOutput is specified	NUMBER
Clustering	CLUSTER_PROBABILITY	only if clusteringProbOutput is specified	BINARY_DOUBLE
Clustering	CLUSTER_SET	only if clusteringProbOutput is specified	Set of (NUMBER, BINARY_DOUBLE)
Clustering	CLUSTER_DISTANCE	only if clusteringDistanceOutput is specified	BINARY_DOUBLE

> **Note**
>
> You can define the outputs explicitly in the metadata or implicitly.
>
> - The metadata must explicitly specify how to find the result in the model output for some SQL scoring functions. For example, `CLUSTER_PROBABILITY` is supported only if `clusteringProbOutput` is specified in the metadata.
> - The system automatically assumes the output for a model with only one output if you don't specify it in the metadata.
> - If a scoring function does not comply according to the description provided, you will receive an ORA-40290 error when performing the scoring operation on your data. Additionally, any unsupported scoring functions will raise the ORA-40290 error.

To learn more about classification data types that are returned, see `labels` and `classificationLabelOutput` in JSON Metadata Parameters for ONNX Models.

Cost Matrix Clause

Specify a cost matrix directly within the `PREDICTION` and `PREDICTION_SET` scoring functions. To learn more about Cost Matrix, see *Oracle Machine Learning for SQL Concepts*.

7.8 Examples of Using ONNX Models

The following examples use the Iris data set to showcase loading and inference from ONNX format machine learning models for machine learning techniques such as Classification, Regression, and Clustering in your Oracle AI Database instance.

Iris is a flower and this data set has information such as petal length, sepal length, petal width, and sepal width collected from three types of Iris flowers: sentosa, versicolour, and virginica.

These examples assume that the data set is available to the user.

ONNX Classification Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX models.

Example: Specifying JSON Metadata for Classification Models

The following example illustrates JSON metadata parameters with Classification as the function. Assume the model has an output named `probabilities` for the probability of the prediction. To use the `PREDICTION_PROBABILITY` scoring function, you must set the field `classificationProbOutput` to the name of the model output that holds the probability.

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities"}'));
END;
/
```

Example: Specifying labels in JSON Metadata for Classification Models

The following example illustrates how you can specify custom labels in the JSON metadata.

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "labels": ["Setosa", "Versicolour", "Virginica"]}'));
END;
/
```

You can use the `PREDICTION` and `PREDICTION_PROBABILITY` functions for inference or scoring:

```
SELECT
    iris.*,
    PREDICTION(doc_model USING *) as predicted_species_id,
    PREDICTION_PROBABILITY(doc_model, 'setosa' USING *) as setosa_probability
FROM iris;
```

The query predicts `iris` species and the probability of *setosa* species using the `iris` data set. The data from `iris` table is used in a `SELECT` query to predict a species ID and the probability that the species is *setosa* using a machine learning model named `doc_model`. The `PREDICTION` function predicts the species based on the attributes in the table, and the `PREDICTION_PROBABILITY` function computes the probability that the predicted species is *setosa*. The result includes all columns from the `iris` view along with the predicted species ID and the probability of the species being *setosa*.

Example: Specifying input in JSON Metadata for Classification Models

The following example illustrates how you can specify input attribute names that map to the actual ONNX model input names. This example assumes a model with four inputs named `SEPAL_LENGTH`, `SEPAL_WIDTH`, `PETAL_LENGTH`, and `PETAL_WIDTH`. You can specify alternative input attribute names using the JSON metadata as shown in this example. Here, each input is assumed to be a tensor with a dimension of 1. The `input` field must be a JSON object where each field is a model input name (For example, `SEPAL_LENGTH`), and its value is a JSON array sized according to the tensor's dimension (here, 1) with one attribute name per element in the array.

```
BEGIN DBMS_VECTOR.LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "input": { "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"],
                  "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                  "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                  "PETAL_WIDTH": ["PETAL_WIDTH_CM"] } }'));
END;
/
```

You can also have a different order of the columns as input.

```
BEGIN DBMS_VECTOR.LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
    JSON('{"function" : "classification",
```

```
            "classificationProbOutput": "probabilities",
            "input": { "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                    "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                    "PETAL_WIDTH": ["PETAL_WIDTH_CM"],
                 "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"] } }'));
END;
/
```

Example: Specifying a Single input With Four Dimensions

Here is an example where the model has a single input tensor named `x` with four dimensions. The corresponding JSON metadata for this scenario is:

```
JSON('{"function" : "classification",
       "classificationProbOutput": "probabilities",
       "input": { "x": ["SEPAL_LENGTH_CM",
                        "SEPAL_WIDTH_CM",
                        "PETAL_LENGTH_CM",
                        "PETAL_WIDTH_CM"]
                  }'));
```

You can use `PREDICTION` and `PREDICTION_PROBABILITY` functions for inference or scoring.

```
WITH
dummy_iris AS (
    SELECT
    4.5 as petal_length_cm,
    1.5 as petal_width_cm,
    4.3 as sepal_length_cm,
    2.9 as sepal_width_cm
    FROM iris
)
SELECT
    dummy_iris.*,
    PREDICTION(doc_model USING *) as predicted_species_id,
    PREDICTION_PROBABILITY(doc_model 'setosa' USING *) as setosa_probability
FROM dummy_iris;
```

The query predicts `iris` species and the probability of *setosa* species using specified attributes in a temporary data set. The query creates a temporary `dummy_iris` view with attributes values set. This temporary view is then used in a `SELECT` query to predict a species ID and the probability that the species is *setosa* using a machine learning model named `doc_model`. The `PREDICTION` function predicts the species based on the attributes provided, and the `PREDICTION_PROBABILITY` function computes the probability that the predicted species is *setosa*. The result includes all columns from the `dummy_iris` view along with the predicted species ID and the probability of the species being *setosa*.

Example: Specifying defaultOnNull in JSON Metadata for Classification Models

The following examples illustrates how you can specify `defaulOnNull` provides default values to be used for specific attributes when their values are NULL in the data set. Use the names `SEPAL_LENGTH`, `SEPAL_WIDTH`, `PETAL_LENGTH`, and `PETAL_WIDTH` as fields in the `defaultOnNull` object, which are the assumed input attribute names for a ONNX model with four inputs. These

names serve as the default input attribute names, so you can use them as fields in the `defaultOnNull`.

```
BEGIN DBMS_VECTOR.LOAD_ONNX_MODEL('classification_model.onnx', 'doc_model',
    JSON('{"function" : "classification",
           "classificationProbOutput": "probabilities",
           "defaultOnNull": {"SEPAL_LENGTH": "5.1",
           "SEPAL_WIDTH": "3.5",
           "PETAL_LENGTH": "1.4",
           "PETAL_WIDTH": "0.2"}}'));
END;
/
```

- `"SEPAL_LENGTH": "5.1"`: If the sepal length is null, use 5.1 as the default value.
- `"SEPAL_WIDTH": "3.5"`: If the sepal width is null, use 3.5 as the default value.
- `"PETAL_LENGTH": "1.4"`: If the petal length is null, use 1.4 as the default value.
- `"PETAL_WIDTH": "0.2"`: If the petal width is null, use 0.2 as the default value.

Example: Specifying input and defaultOnNull JSON Metadata for Classification Models

Here is a combined example of specifying `input` and `defaultOnNull` values. This example uses the values that were illustrated in the earlier examples where `input` and `defaultOnNull` values are specified:

```
JSON('{"function" : "classification",
               "classificationProbOutput": "probabilities",
               "input": { "SEPAL_WIDTH": ["SEPAL_WIDTH_CM"],
                          "PETAL_LENGTH": ["PETAL_LENGTH_CM"],
                          "PETAL_WIDTH": ["PETAL_WIDTH_CM"],
                          "SEPAL_LENGTH": ["SEPAL_LENGTH_CM"] },
                    "defaultOnNull": {"SEPAL_LENGTH_CM": "5.1",
                                      "SEPAL_WIDTH_CM": "3.5"}}')
```

ONNX Clustering Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX models.

Example: Specifying JSON Metadata for Clustering Models

The following example illustrates JSON metadata parameters with Clustering as the function. Assume the model has an output named `probabilities` for the probability of the prediction. To use the `CLUSTER_PROBABILITY` scoring function, you must set the field `clusteringProbOutput` to the name of the model output that holds the probability.

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('clustering_model.onnx','doc_model',
    JSON('{"function": "clustering",
           "clusteringProbOutput": "probabilities"
        }
    ')
);
```

```
END;
/
```

You can use `CLUSTER_ID` and `CLUSTER_PROBABILITY` functions for inference or scoring.

```
SELECT
    iris.*,
    CLUSTER_ID(doc_model USING *) as cluster_id,
    CLUSTER_PROBABILITY(doc_model, 1 USING *) as cluster_1_probability
FROM iris;
```

This query predicts the cluster assignments and the probabilities of belonging to a specific cluster for each record of the `iris` data set. The query retrieves all columns of each record (`iris.*`) and applies the clustering model named `doc_model` to each record of the `iris` data set and predicts the cluster ID. The `USING *` clause tells the model to use all available columns in the `iris` table for this prediction. The `CLUSTER_PROBABILITY(doc_model, 1 USING *) as cluster_1_probability` part of the query calculates the probability that each record belongs to cluster 1, according to the `doc_model` from the `iris` data set. This provides insights into how likely each record is to be part of cluster 1, giving a quantitative measure of membership strength.

Example: Specifying clusteringDistanceOutput in JSON Metadata for Clustering Models

The following example illustrates how you can specify `clusteringDistanceOutput` and for ONNX Clustering models.

In this model, an output tensor named `distances` provides distances for the input, which is a single tensor named `float_input` with a dimension of 4. The JSON metadata `input` field must map attribute names to entries of the tensor, such as `"SEPAL_LENGTH", "SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH"`.

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('clustering_model.onnx', 'doc_model',
JSON('{"function" : "clustering",
"clusteringDistanceOutput": "distances",
"normalizeProb": "softmax",
"input": { "float_input": ["SEPAL_LENGTH", "SEPAL_WIDTH", "PETAL_LENGTH",
"PETAL_WIDTH"] }
      }')
);
END;
/
```

You can use `CLUSTER_DISTANCE` function for inference or scoring. These SQL queries utilize clustering models to predict cluster distances from the `IRIS` data set.

```
SELECT CLUSTER_DISTANCE(doc_model USING *) AS predicted_target_value,
CLUSTER_DISTANCE (doc_model,1 USING *) AS dist1,
CLUSTER_DISTANCE (doc_model,2 USING *) AS dist2,
CLUSTER_DISTANCE (doc_model,3 USING *) AS dist3
FROM IRIS
ORDER BY ID
FETCH NEXT 10 ROWS ONLY;
```

Here, the query focuses on understanding the physical distance of data points from cluster centroids, which is particularly useful for identifying outliers or for performing detailed cluster analysis. The query calculates the distance of each record in the `IRIS` data set from the centroids of different clusters using the `doc_model`. The `USING *` syntax indicates that the model must use all available columns of the `IRIS` data set for making the prediction. `CLUSTER_DISTANCE(doc_model, n USING *)` computes the distance from cluster `n` (`n` being 1, 2, and 3 in this query). Each distance is selected as a separate column (`dist1`, `dist2`, `dist3`).

The output is limited to the first 10 rows of the result set ordered by the `ID` column of the `IRIS` table.

Example: Specifying clusteringProbOutput and normalizeProb in JSON Metadata for Clustering Models

The following example illustrates how you can specify `clusteringProbOutput` and `normalizeProb` for ONNX Clustering models.

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('clustering_model.onnx', 'doc_model',
               JSON('{"function" :
               "clustering",
               "clusteringProbOutput": "probabilities",
                        "normalizeProb" : "softmax",
               "input": { "float_input": ["SEPAL_LENGTH", "SEPAL_WIDTH",
"PETAL_LENGTH", "PETAL_WIDTH"] } }')
      );
END;
/
```

You can use `CLUSTER_PROBABILITY` and `CLUSTER_SET` functions for inference or scoring:

```
SELECT CLUSTER_ID (doc_model USING *) AS predicted_target_value,
                    CLUSTER_PROBABILITY (doc_model,1 USING *) AS prob1,
                    CLUSTER_PROBABILITY (doc_model,2 USING *) AS prob2,
                    CLUSTER_PROBABILITY (doc_model,3 USING *) AS prob3
                    FROM IRIS
                    ORDER BY ID
                    FETCH NEXT 10 ROWS ONLY;
```

In this case, a clustering model is used to predict the cluster IDs and associated probabilities for records from the `IRIS` data set. Because the JSON metadata specifies `softmax` for the `normalizeProb` field, the model applies softmax normalization to the probabilities before returning them as the result of the `CLUSTER_PROBABILITY` scoring operator.

The SQL query selects `CLUSTER_ID` column from the `IRIS` table and adds a new column, `predicted_target_value`, which contains predictions made by the `doc_model`. The `USING *` syntax means that all columns of the current row are used as input features for the `doc_model` model to predict the value as `predicted_target_value`. The result of this prediction is then included as a new column in the output of the query.

`CLUSTER_PROBABILITY(model, n USING *)`: Computes the probability that the record belongs to cluster `n` (`n` being 1, 2, and 3 in this query). This is done for three different clusters, and each probability is selected as a separate column (`prob1`, `prob2`, `prob3`).

The output is limited to the first 10 rows of the result set ordered by the `ID` column of the `IRIS` table.

```
SELECT S.CLUSTER_ID, S.PROBABILITY
 FROM (SELECT CLUSTER_SET(doc_model USING *) pset
          FROM IRIS ORDER BY ID) T,
       TABLE(T.pset) S
FETCH NEXT 10 ROWS ONLY;
```

The `CLUSTER_SET` query generates a set of cluster data using the `doc_model`. The resultant column `pset` represents all possible cluster assignments for each record, which includes cluster IDs and their respective probabilities ordered by the `ID` column. The `SELECT S.CLUSTER_ID, S.PROBABILITY` part of the query selects the cluster ID and probability from the resultant column set. The output is limited to the first 10 rows of the result set.

ONNX Regression Examples

The following examples showcase various JSON metadata parameters that can be defined for ONNX Regression models. All examples assume an ONNX model that has one output named `regressionOutput` and four input tensors of dimension 1 whose name match exactly the name of the `IRIS` table columns, namely, `SEPAL_LENGTH, SEPAL_WIDTH, PETAL_LENGTH, PETAL_WIDTH`.

Example: Specifying JSON Metadata for Regression Models

The following is a simple example illustrating JSON metadata parameters with Regression as the function. Assume the ONNX model features one output named `regressionOutput` and four input tensors of dimension 1, whose names match exactly after the `IRIS` table columns (`"SEPAL_LENGTH", "SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH"`). The JSON metadata can be as simple as the following:

```
BEGIN DBMS_VECTOR.LOAD_ONNX_MODEL(
    'regression_model.onnx',
    'doc_model',
    JSON('{"function": "regression"}
    ')
);
END;
/
```

You can use the `PREDICTION` function for inference or scoring:

```
SELECT
    iris.*,
    PREDICTION(doc_model USING *) as predicted_petal_width_cm
FROM iris;
```

In this case, the SQL query selects all columns from the `iris` table and adds a new column, `predicted_petal_width_cm`, which contains predictions made by the `doc_model`. The `USING *` syntax means that all columns of the current row are used as input features for the `doc_model` model to predict the value of `PETAL_WIDTH` as `predicted_petal_width_cm`. The result of this prediction is then included as a new column in the output of the query.

Example: Specifying input and defaultOnNull in JSON Metadata for Regression Models

The following example illustrates how you can specify input attribute names that map to the actual ONNX model input names. The `defaulOnNull` providing default values to be used for specific attributes when their values are NULL in the data set.

```
BEGIN DBMS_VECTOR.LOAD_ONNX_MODEL('regression_model.onnx','doc_model',
    JSON('{"function": "regression",
            "input": {
                "SEPAL_LENGTH": ["dummy_sepal_length_cm"],
                "SEPAL_WIDTH": ["dummy_sepal_width_cm"]
                      },
            "defaultOnNull": {
                "dummy_sepal_length_cm": "5.1",
                "dummy_sepal_width_cm": "3.5",
                 }
          }
        ')
);
END;
/
```

You can use the `PREDICTION` function for inference or scoring.

```
WITH
dummy_iris AS (
    SELECT
    (CASE WHEN petal_length > 5 THEN 4.9 ELSE NULL END)
        as dummy_sepal_length_cm,
    (CASE WHEN petal_length < 4 THEN 2.5 ELSE NULL END)
        as dummy_sepal_width_cm,
    petal_length
    petal_width
    FROM iris
)
SELECT
    dummy_iris.*,
    PREDICTION(doc_model USING *) as predicted_petal_width_cm
FROM dummy_iris;
```

In this case, a temporary `dummy_iris` table is created with three columns: `dummy_sepal_length_cm`, `dummy_sepal_width_cm`, and `petal_length`. The values of the `dummy_sepal_length_cm` and `dummy_sepal_width_cm` are based on `petal_length` values of the `iris` table. If `petal_length` is greater than 5, `dummy_sepal_length_cm` is set to 4.9, otherwise it is NULL. If `petal_length` is less than 4, `dummy_sepal_width_cm` is set to 2.5, otherwise it remains NULL.

Then the `SELECT` query retrieves all columns from the `dummy_iris` table and uses the `doc_model` to predict `petal_width`, adding this prediction as a new column named `predicted_petal_width_cm`. The model uses the derived dummy columns, `petal_length` and `petal_width` for its predictions.

See Also

- LOAD_ONNX_MODEL in *Oracle Database PL/SQL Packages and Types Reference*
- Supported SQL Scoring Functions

7.9 Traditional Machine Learning ONNX Format Models

Traditional machine learning models using algorithms such as decision trees, random forests, and support vector machines, among others, can be converted to ONNX format. Such models may be produced in other environments and deployed through Oracle AI Database.

Once such models are converted to ONNX format, they can be deployed directly in Oracle AI Database and use the ONNX Runtime for inference through the SQL prediction operators. These models are typically used for tasks such as Classification, Regression, and Clustering.

Related Topics

- Examples of Using ONNX Models
  The following examples use the Iris data set to showcase loading and inference from ONNX format machine learning models for machine learning techniques such as Classification, Regression, and Clustering in your Oracle AI Database instance.

7.10 Text Transformer ONNX Format Models

Text transformers have the ability to translate natural language text into a numerical vector representation also known as an embedding, you use such vectors for semantic similarity search or other Natural Language Processing (NLP) use cases.

Models such as BERT, sentence transformer models from Hugging Face, and other transformer-based models can be converted into ONNX format models. These models can be run within Oracle AI Database. These models can be used in AI vector search within Oracle AI Database, where documents are compared based on their mathematical distance between the vectors to determine the similarity.

Related Topics

- Examples of Using ONNX Models
  The following examples use the Iris data set to showcase loading and inference from ONNX format machine learning models for machine learning techniques such as Classification, Regression, and Clustering in your Oracle AI Database instance.

7.11 Image Transformer ONNX Format Models

Image transformer is a part of machine learning that helps computers interpret and analyze images and videos. It provides tools to perform tasks like creating image embeddings (using an image transformer), classifying objects, detecting anomalies, and identifying objects in pictures or videos.

Image transformers don't directly use images as input. They need pre-processing to convert images into a form the model can understand. Common pre-processing steps include:

- Decoding images from formats like JPEG to a 3D numeric array.
- Resizing images to standard dimensions.

- Normalizing pixel values.
- Reducing noise in the image.
- Cropping parts of the image for focus.

Image transformer models can be converted into the ONNX format and used directly in Oracle AI Database. Each image transformer requires its own specific pre-processing pipeline and Oracle offers OML4Py pre-processing pipeline for such models.

- Pretrained Image Transformer Models in Oracle AI Database
  Oracle AI Database supports using pretrained image transformer models for generating vectors for semantic similarity search.
- Example: Generate Embeddings from Image Transformer Models
  The following examples illustrate generating embeddings from images with image transformer model using `DBMS_VECTOR` or `DBMS_DATA_MINING` packages and use the ONNX Runtime for inference through the SQL prediction operators.

7.11.1 Pretrained Image Transformer Models in Oracle AI Database

Oracle AI Database supports using pretrained image transformer models for generating vectors for semantic similarity search.

You can access image transformer models through machine learning platforms like Hugging Face that provide pretrained models for immediate use.

To use pretrained image transformer models in Oracle AI Database, here are the high-level steps:

- Download pretrained models: Download image transformer models into the database.
- Convert image transformer model to ONNX format: Use ONNX pipeline to convert the pretrained image transformer model to ONNX format. Add image pre-processing by implementing Oracle's custom ONNX operation for image decoding and create a model-specific ONNX pre-processing pipeline. See Import Pretrained Models in ONNX Format for Vector Generation Within the Database for more details.
- Import ONNX format image transformer model: Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure or `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` to import the ONNX model into your Oracle AI Database. After importing, use the `VECTOR_EMBEDDING` operator to generate vector embeddings from JPEG images stored as BLOB in the database.

> **Note**
>
> Only JPEG images are supported. Multiple ONNX models may have to be loaded for multi-modal model because each modality has a different pre-processing and post-processing pipeline.

The Oracle AI Database supports popular pretrained models such as:

- ResNet-50: A widely used model for image classification.
- CLIP ViT-Base-Patch32: A multi-modal model for linking text and image content.
- ViT Base-Patch: A vision transformer model designed for image analysis and classification.

7.11.2 Example: Generate Embeddings from Image Transformer Models

The following examples illustrate generating embeddings from images with image transformer model using `DBMS_VECTOR` or `DBMS_DATA_MINING` packages and use the ONNX Runtime for inference through the SQL prediction operators.

These examples assume that:

- the data set is available to the user.
- the `DM_DUMP` directory exists and contains the ONNX model file for image transformer models augmented with image pre-processing. Follow the steps in ONNX Pipeline Models: Image Embedding and ONNX Pipeline Models: Multi-modal Embedding to generate the ONNX files for the ResNet-50 and Clip ViT models. See also Import ONNX Models into Oracle AI Database End-to-End Example.

Load File Contents into a BLOB
The following example loads the contents of a file stored in a directory object (`DM_DUMP`) into a BLOB in the database. The function returns the `BLOB` containing the file content.

```
create or replace
    function loader(p_filename varchar2) return blob is
      bf bfile := bfilename('DM_DUMP',p_filename);
      b blob;
    begin
      dbms_lob.createtemporary(b,true);
      dbms_lob.fileopen(bf, dbms_lob.file_readonly);
      dbms_lob.loadfromfile(b,bf,dbms_lob.getlength(bf));
      dbms_lob.fileclose(bf);
     return b;
   end;
   /
```

Create image_data Table
The following example creates the `image_data` table assuming image files are under the `DM_DUMP` directory. The `image_data` table is used further for generating vector embeddings.

```
SQL> CREATE TABLE image_data (
      ID NUMBER,
      NAME VARCHAR2(20),
      IMAGE BLOB
    );

Table created.


SQL> insert into image_data values (1,'cat.jpg',loader('cat.jpg'));

1 row created.

SQL> insert into image_data values (2,'cat2.jpg',loader('cat2.jpg'));

1 row created.

SQL> insert into image_data values (3,'chicken.jpg',loader('chicken.jpg'));
```

```
1 row created.

SQL> insert into image_data values (4,'horse.jpg',loader('horse.jpg'));

1 row created.

SQL> insert into image_data values (5,'dog.jpg',loader('dog.jpg'));

1 row created.

SQL> insert into image_data values (6,'cat.png',loader('cat.png'));

1 row created.

SQL> commit;

Commit complete.
```

Load a ResNet-50 Computer Image Transformer and Generate Vector Embeddings
The following example demonstrates loading an image tranformer model extended with image pre-processing pipeline and using it to generate vector embeddings from images stored in a BLOB. The example assumes that the `DM_DUMP` directory exists and contains the ONNX file for ResNet-50 model augmented with ONNX-based image pre-processing pipeline.

The example imports a pretrained ONNX-format transformer model (`pp_resnet_50.onnx`) into the database as `ppresnet50` using the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure. Alternately, you can load the model using the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL`. After checking the dictionary views, and examining the schema of the `image_data` table, the model runs a query that generates vector embeddings for each image stored in the `image_data` table using the `VECTOR_EMBEDDING` operator. The vector embeddings can be further used for image classification, similarity search, or feature extraction. The query returns the first 40 characters of each vector. For unsupported formats such as `cat.png` (a PNG file), the `VECTOR_EMBEDDING` operator returns a NULL value.

```
-- Metadata for an embedding model
SQL> define ppjsonmd = '{"function" : "embedding"}';

SQL> exec DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP', 'pp_resnet_50.onnx',
'ppresnet50');

PL/SQL procedure successfully completed.


SQL> SELECT mining_function, algorithm, model_size FROM user_mining_models
WHERE model_name = 'PPRESNET50';

MINING_FUNCTION               ALGORITHM            MODEL_SIZE
----------------------------- -------------------- ----------
EMBEDDING                     ONNX                 93979933


SQL> SELECT attribute_name, attribute_type, data_type, vector_info FROM
user_mining_model_attributes WHERE model_name = 'PPRESNET50' ORDER BY 1;
```

```
ATTRIBUTE_NAME              ATTRIBUTE_TYPE       DATA_TYPE  VECTOR_INFO
--------------------    --------------------  ----------
--------------------
DATA                         UNSTRUCTURED         BLOB
ORA$ONNXTARGET               VECTOR               VECTOR
VECTOR(2048,FLOAT32)


SQL> describe image_data

Name
                                          Null?    Type
 ----------------------------------------------------------------- --------
----------------------------------------

ID
                                                                   NUMBER

NAME
                                                                   VARCHAR2(20)

IMAGE
                                                                   BLOB


SQL> SELECT name, substr(vector_embedding(ppresnet50 using image as data), 0,
40) as vec FROM image_data;

NAME                      VEC
-------------------- ----------------------------------------
cat.jpg              [0,3.69947255E-002,1.727576E-002,0,6.437
cat2.jpg             [5.25364205E-002,0,0,2.8940714E-003,0,4.
chicken.jpg          [2.14146048E-001,7.94866239E-004,2.95593
horse.jpg            [1.63398478E-002,0,4.99145657E-001,0,0,1
dog.jpg              [0,0,7.96773005E-004,0,0,0,1.00504747E-0
cat.png

6 rows selected.
```

Alternately, use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure to import the `ppresnet50` model into the database and proceed with the rest of the steps as shown in the example. Here, the loader function loads the content of the file or ONNX files into a blob.

```
SQL> exec DBMS_DATA_MINING.IMPORT_ONNX_MODEL('ppresnet50',
loader('pp_resnet_50.onnx'), JSON('&ppjsonmd'));

PL/SQL procedure successfully completed.
```

Load a CLIP ViT Model to Generate Vector Embeddings from Images (Image Modality) and Search Images by Generating Embedding from Text Description (Text Modality)
The following example uses CLIP ViT Base patch model (`ppclip`) to check pre-configured ONNX-based image embedding pipeline and generates vector embeddings. The example assumes that the `DM_DUMP` directory exists and contains the ONNX files for each modality of the CLIP ViT Base patch model. The `pp_clip_img.onnx` holds the model augmented with ONNX-based image pre-processing and post-processing pipelines needed for image modality. The

`pp_clip_txt.onnx` holds the model augmented with ONNX-based pre-processing and post-processing pipelines for text modality. Follows the steps in ONNX Pipeline Models: Multi-modal Embedding to get the ONNX files for each of the modality of the CLIP ViT Base patch model.

```
SQL> set echo on
SQL> -- Import clip model with image preprocessing (image modality)
SQL> exec DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP', 'pp_clip_img.onnx',
'clipimg');

PL/SQL procedure successfully completed.

SQL> -- Import clip model with text preprocessing (text modality)
SQL> exec DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP', 'pp_clip_txt.onnx',
'cliptxt');

PL/SQL procedure successfully completed.

SQL> -- Show difference between the two modality:
SQL> SELECT model_name, attribute_name, attribute_type, data_type,
vector_info FROM user_mining_model_attributes WHERE model_name LIKE 'CLIP%'
ORDER BY 1,2;


MODEL_NAME     ATTRIBUTE_NAME      ATTRIBUTE_TY        DATA_TYPE
VECTOR_INFO
----------     --------------- ------------       ----------------
--------------------
CLIPIMG     DATA             UNSTRUCTURED           BLOB
CLIPIMG     ORA$ONNXTARGET         VECTOR                VECTOR
VECTOR(512,FLOAT32)
CLIPTXT     DATA             TEXT            VARCHAR2
CLIPTXT     ORA$ONNXTARGET         VECTOR                VECTOR
VECTOR(512,FLOAT32)


SQL> -- Create a table with vectors generated from image using clip
SQL> CREATE TABLE image_vectors as select name, vector_embedding(clipimg
using image as data) as embedding FROM image_data;

Table created.

SQL> -- Find top-3 similar image from text description
SQL> select name from image_vectors order by
vector_distance(vector_embedding(cliptxt using 'Cat picture' as data),
embedding) fetch first 2 rows only;

NAME
--------------------
cat.jpg
cat2.jpg
```

Alternately, use the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure to load the `clipimg` and `cliptxt` models into the database.

```
-- Import CLIP model with image preprocessing (image modality)
SQL> exec DBMS_DATA_MINING.IMPORT_ONNX_MODEL('clipimg',
loader('pp_clip_img.onnx'), JSON('{"function" : "embedding"}'));

PL/SQL procedure successfully completed.

-- Import CLIP model with text preprocessing (text modality)
SQL> exec DBMS_DATA_MINING.IMPORT_ONNX_MODEL('cliptxt',
loader('pp_clip_txt.onnx'), JSON('{"function" : "embedding"}'));

PL/SQL procedure successfully completed.
```

8

Administrative Tasks for Oracle Machine Learning for SQL

Explains how to perform administrative tasks related to Oracle Machine Learning for SQL.

- Install and Configure a Database for Oracle Machine Learning for SQL
  You can install and configure a database for Oracle Machine Learning for SQL by following the listed steps.
- Upgrade or Downgrade Oracle Machine Learning for SQL
  Upgrade and downgrade Oracle Machine Learning for SQL by following the steps listed.
- Export and Import Oracle Machine Learning for SQL Models
  You can export machine learning models to move models to a different Oracle AI Database instance, such as from a development database to a production database.
- Control Access to Oracle Machine Learning for SQL Models and Data
  You can create a Oracle Machine Learning for SQL user and grant necessary privileges by following the steps listed.
- Audit and Add Comments to Oracle Machine Learning for SQL Models
  Perform audit of Oracle Machine Learning for SQL model objects through SQL statements.

8.1 Install and Configure a Database for Oracle Machine Learning for SQL

You can install and configure a database for Oracle Machine Learning for SQL by following the listed steps.

- About Installation
  Oracle Machine Learning components associated with Oracle AI Database are included with the database license.
- Database Tuning Considerations for Oracle Machine Learning for SQL
  Standard administrative practices can be followed to manage workload on the system when machine learning activities are running.

8.1.1 About Installation

Oracle Machine Learning components associated with Oracle AI Database are included with the database license.

To install Oracle AI Database, follow the installation instructions for your platform. Choose a Data Warehousing configuration during the installation.

Oracle Data Miner, the graphical user interface to Oracle Machine Learning for SQL, is an extension to Oracle SQL Developer. Instructions for downloading SQL Developer and installing the Data Miner repository are available on https://www.oracle.com/database/technologies/odmrinstallation.html.

To perform machine learning activities, you must be able to log on to the Oracle Database, and your user ID must have the database privileges described in Grant Privileges for Oracle Machine Learning for SQL.

Related Topics

- Oracle Data Miner

> **See Also**
>
> **Install and Upgrade** page of the Oracle AI Database online documentation library for your platform-specific installation instructions: `Oracle AI Database 26ai Release`

8.1.2 Database Tuning Considerations for Oracle Machine Learning for SQL

Standard administrative practices can be followed to manage workload on the system when machine learning activities are running.

DBAs managing production databases that support Oracle Machine Learning for SQL must follow standard administrative practices as described in *Oracle Database Administrator's Guide*.

Building machine learning models and batch scoring of machine learning models tend to put a DSS-like workload on the system. Single-row scoring tends to put an OLTP-like workload on the system.

Database memory management can have a major impact on machine learning. The correct sizing of Program Global Area (PGA) memory is very important for model building, complex queries, and batch scoring. From a machine learning perspective, the System Global Area (SGA) is generally less of a concern. However, the SGA must be sized to accommodate real-time scoring, which loads models into the shared cursor in the SGA. In most cases, you can configure the database to manage memory automatically. To do so, specify the total maximum memory size in the tuning parameter `MEMORY_TARGET`. With automatic memory management, Oracle AI Database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands.

Most machine learning algorithms can take advantage of parallel execution when it is enabled in the database. Parameters in `INIT.ORA` control the behavior of parallel execution.

8.2 Upgrade or Downgrade Oracle Machine Learning for SQL

Upgrade and downgrade Oracle Machine Learning for SQL by following the steps listed.

- Pre-Upgrade Steps
  Pre-upgrade considerations.
- Upgrade Oracle Machine Learning for SQL
  You can upgrade your database by using the Database Upgrade Assistant (DBUA) or you can perform a manual upgrade using export/import utilities.
- Post Upgrade Steps
  Perform steps to view the upgraded database.

- Downgrade Oracle Machine Learning for SQL
  Before downgrading the Oracle AI Database back to the previous version, ensure that no models are present.

8.2.1 Pre-Upgrade Steps

Pre-upgrade considerations.

Before upgrading, you must drop any machine learning models and machine learning activities that were created inOracle Data Miner.

8.2.2 Upgrade Oracle Machine Learning for SQL

You can upgrade your database by using the Database Upgrade Assistant (DBUA) or you can perform a manual upgrade using export/import utilities.

All models and machine learning metadata are fully integrated with the Oracle AI Database upgrade process whether you are upgrading from 19*c* or from earlier releases.

Upgraded models continue to work as they did in prior releases. Both upgraded models and new models that you create in the upgraded environment can make use of the new machine learning functionality introduced in the new release.

- Use Database Upgrade Assistant to Upgrade Oracle Machine Learning for SQL
  Oracle AI Database Upgrade Assistant provides a graphical user interface that guides you interactively through the upgrade process.
- Use Export/Import to Upgrade Machine Learning Models
  Use Export and Import functions of the Oracle AI Database to export the previously created models and import the models in an instance of Oracle Database version.

Related Topics

- Pre-Upgrade Steps
  Pre-upgrade considerations.
- *Oracle Database Upgrade Guide*

8.2.2.1 Use Database Upgrade Assistant to Upgrade Oracle Machine Learning for SQL

Oracle AI Database Upgrade Assistant provides a graphical user interface that guides you interactively through the upgrade process.

On Windows platforms, follow these steps to start the Upgrade Assistant:

1. Go to the Windows **Start** menu and choose the Oracle home directory.
2. Choose the **Configuration and Migration Tools** menu.
3. Launch the **Upgrade Assistant**.

On Linux platforms, run the `DBUA` utility to upgrade Oracle AI Database.

Related Topics

- *Oracle Database Upgrade Guide*

8.2.2.2 Use Export/Import to Upgrade Machine Learning Models

Use Export and Import functions of the Oracle AI Database to export the previously created models and import the models in an instance of Oracle Database version.

If required, you can use a less automated approach to upgrading machine learning models. You can export the models created in a previous version of Oracle Database and import them into an instance of the Oracle AI Database version.

- Export/Import Oracle Machine Learning for SQL Models
  Use the export and import functions of the Oracle AI Database to export the previously created models and import the models in an instance of Oracle AI Database version.

8.2.2.2.1 Export/Import Oracle Machine Learning for SQL Models

Use the export and import functions of the Oracle AI Database to export the previously created models and import the models in an instance of Oracle AI Database version.

If required, you can use a less automated approach to upgrading machine learning models. You can export the models created in a previous version of Oracle Database and import them into an instance of the Oracle AI Database version.

To export models from an instance of a previous release of Oracle AI Database to a dump file, follow the instructions in Export and Import Oracle Machine Learning for SQL Models.

8.2.3 Post Upgrade Steps

Perform steps to view the upgraded database.

After upgrading the database, check the `DBA_MINING_MODELS` view in the upgraded database. The newly upgraded machine learning models must be listed in this view.

After you have verified the upgrade and confirmed that there is no need to downgrade, you must set the initialization parameter `COMPATIBLE` to `23.6.0`. In Oracle AI Database 26ai, when the `COMPATIBLE` initialization parameter is not set in your parameter file, the `COMPATIBLE` parameter value defaults to `23.6.0`.

> **Note**
>
> The `CREATE MINING MODEL` privilege must be granted to Oracle Machine Learning for SQL user accounts that are used to create machine learning models.

Related Topics

- Create an Oracle Machine Learning for SQL User
  An OML4SQL user is a database user account that has privileges for performing machine learning activities.
- Control Access to Oracle Machine Learning for SQL Models and Data
  You can create a Oracle Machine Learning for SQL user and grant necessary privileges by following the steps listed.

8.2.4 Downgrade Oracle Machine Learning for SQL

Before downgrading the Oracle AI Database back to the previous version, ensure that no models are present.

Use the `DBMS_DATA_MINING.DROP_MODEL` routine to drop the models before downgrading. If you do not do this, the database downgrade process terminates.

Issue the following SQL statement in `SYS` to verify the downgrade:

```
SELECT o.name FROM sys.model$ m, sys.obj$ o
                 WHERE m.obj#=o.obj# AND m.version=2;
```

8.3 Export and Import Oracle Machine Learning for SQL Models

You can export machine learning models to move models to a different Oracle AI Database instance, such as from a development database to a production database.

The `DBMS_DATA_MINING` package includes procedures for migrating machine learning models between database instances.

`EXPORT_MODEL` exports a single model or list of models to a dump file so it can be imported, queried, and scored in a separate Oracle Machine Learning database instance.

`IMPORT_MODEL` takes the dump file and creates the model in the destination database.

`EXPORT_SERMODEL` exports a single model to a serialized `BLOB` so it can be imported and scored in a separate Oracle Machine Learning database instance or to OML Services.

`IMPORT_SERMODEL` takes the serialized `BLOB` and creates the model in the destination database.

- About Exporting Models
  A mining model is a first-class object with dependent tables and views. When you drop or rename a mining model, your database handles the dependent objects internally.
- About Oracle Data Pump
  Use the command-line clients of Oracle Data Pump to export and import schemas or databases.
- Options for Exporting and Importing Oracle Machine Learning for SQL Models
  Lists options for exporting and importing machine learning models.
- Directory Objects for EXPORT_MODEL and IMPORT_MODEL
  Learn how to use directory objects to identify the location of the dump file set containing the models.
- Use EXPORT_MODEL and IMPORT_MODEL
  The examples illustrate various export and import scenarios with `EXPORT_MODEL` and `IMPORT_MODEL`.
- EXPORT and IMPORT Serialized Models
  From Oracle Database Release 18c onwards, `EXPORT_SERMODEL` and `IMPORT_SERMODEL` procedures are available to export or import serialized models to or from a database.
- Import From PMML
  You can import regression models represented in Predictive Model Markup Language (PMML).

Related Topics

- `EXPORT_MODEL`

- IMPORT_MODEL
- EXPORT_SERMODEL
- IMPORT_SERMODEL

8.3.1 About Exporting Models

A mining model is a first-class object with dependent tables and views. When you drop or rename a mining model, your database handles the dependent objects internally.

Data Pump exports the mining model along with all dependent objects, either as part of a schema-level export or as an individual model export. During import, Data Pump recreates the system metadata and the model tables and views.

> **Note**
>
> If you remove any of the dependent objects directly, the model tracks this and reports itself as corrupted. You can only drop a model that is in a corrupted state. Removing dependent objects requires SYS privileges.

For serialized model export using EXPORT_SERMODEL, the database stores the model in a single internal table that includes only scoring information. It does not maintain any associated model views or additional tables. Serialized model export only works with models that produce scores. Serialized model export supports models that produce scores, such as Classification, Regression, and Clustering models (including *k*-Means and Expectation Maximization) among others.. Use EXPORT_MODEL to export these models and scenarios when full model details are needed.

The structure and number of dependent objects vary by algorithm. You can see the associated tables and views in the DBA_MINING_MODEL_TABLES and USER/ALL/DBA_MINING_MODEL_VIEWS views.

To retain complete model details, use the DMBS_DATA_MINING.EXPORT_MODEL procedure and the DBMS_DATA_MINING.IMPORT_MODEL procedure.

Related Topics

- EXPORT_MODEL Procedure
- IMPORT_MODEL Procedure

8.3.2 About Oracle Data Pump

Use the command-line clients of Oracle Data Pump to export and import schemas or databases.

Oracle Data Pump consists of two command-line clients and two PL/SQL packages. The command-line clients, expdp and impdp, provide an easy-to-use interface to the Data Pump export and import utilities. You can use expdp and impdp to export and import entire schemas or databases respectively.

The Data Pump export utility writes the schema objects, including the tables and metadata that constitute machine learning models, to a dump file set. The Data Pump import utility retrieves the schema objects, including the model tables and metadata, from the dump file set and restores them in the target database.

When you export mining models using Data Pump, the utility captures both the core model data and the dependent views and metadata. Oracle handles these dependencies automatically during import. For serialized exports, Oracle stores only the core scoring content without related views or auxiliary tables.

`expdp` and `impdp` cannot be used to export/import individual machine learning models.

> **See Also**
>
> *Oracle Database Utilities* for information about Oracle Data Pump and the `expdp` and `impdp` utilities

8.3.3 Options for Exporting and Importing Oracle Machine Learning for SQL Models

Lists options for exporting and importing machine learning models.

Options for exporting and importing machine learning models are described in the following table.

Table 8-1 Export and Import Options for Oracle Machine Learning for SQL

Task	Description
Export or import a full database	(DBA only) Use `expdp` to export a full database and `impdp` to import a full database. All machine learning models in the database are included.
Export or import a schema	Use `expdp` to export a schema and `impdp` to import a schema. All machine learning models in the schema are included.
Export or import models within a database or between databases	Use `DBMS_DATA_MINING.EXPORT_MODEL` to export one or more models and `DBMS_DATA_MINING.IMPORT_MODEL` to import one or more models. These procedures can export and import a single machine learning model, all machine learning models, or machine learning models that match specific criteria. To import models, you must have the `CREATE TABLE`, `CREATE VIEW`, and `CREATE MINING MODEL` privileges.
Export or import individual models to or from a remote database	Use a database link to export individual models to a remote database or import individual models from a remote database. A database link is a schema object in one database that enables access to objects in a different database. The link must be created before you run `EXPORT_MODEL` or `IMPORT_MODEL`. To create a private database link, you must have the `CREATE DATABASE LINK` system privilege. To create a public database link, you must have the `CREATE PUBLIC DATABASE LINK` system privilege. Also, you must have the `CREATE SESSION` system privilege on the remote Oracle AI Database. Oracle Net must be installed on both the local and remote Oracle AI Databases.

Table 8-1 (Cont.) Export and Import Options for Oracle Machine Learning for SQL

Task	Description
Serialized model export and import	Starting from Oracle Database 18*c*, the serialized model format was introduced as a lightweight approach to support scoring. The `DBMS_DATA_MINING.EXPORT_SERMODEL` procedure exports a single model to a serialized `BLOB` so it can be imported and scored in a separate Oracle Machine Learning (OML) database instance or to OML Services. `DBMS_DATA_MINING.IMPORT_SERMODEL` takes the serialized `BLOB` and creates the model in the destination database. **Note** Serialized model export does not maintain additional tables or views. It only includes the model object and metadata required for scoring. Use `DBA_MINING_MODEL_TABLES` and related views to understand model dependencies for non-serialized formats.

Related Topics

- IMPORT_MODEL Procedure
- EXPORT_MODEL Procedure
- *Oracle Database SQL Language Reference*

8.3.4 Directory Objects for EXPORT_MODEL and IMPORT_MODEL

Learn how to use directory objects to identify the location of the dump file set containing the models.

`EXPORT_MODEL` and `IMPORT_MODEL` use a directory object to identify the location of the dump file set. A directory object is a logical name in the database for a physical directory on the host computer.

To export machine learning models, you must have write access to the directory object and to the file system directory that it represents. To import machine learning models, you must have read access to the directory object and to the file system directory. Also, the database itself must have access to file system directory. You must have the `CREATE ANY DIRECTORY` privilege to create directory objects.

The following SQL command creates a directory object named `omldir`. The file system directory that it represents must already exist and have shared read/write access rights granted by the operating system. For example, if the directory path is `/home/omluser_dir`, the command is:

```
CREATE OR REPLACE DIRECTORY omldir AS '/home/omluser_dir';
```

The following SQL command gives user `omluser` both read and write access to `omldir`.

```
GRANT READ,WRITE ON DIRECTORY omldir TO OMLUSER;
```

Related Topics

- *Oracle Database SQL Language Reference*

8.3.5 Use EXPORT_MODEL and IMPORT_MODEL

The examples illustrate various export and import scenarios with `EXPORT_MODEL` and `IMPORT_MODEL`.

The examples use the directory object `OMLDIR` shown in Example 8-1 and two schemas, `DM1` and `DM2`. Both schemas have machine learning privileges. `DM1` has two models. `DM2` has one model.

The `DM1` schema has the following models:

- The `EM_SH_CLUS_SAMPLE` model: it is created by the `oml4sql-clustering-expectation-maximization.sql` example.
- The `DT_SH_CLAS_SAMPLE` model: it is created by the `oml4sql-classification-decision-tree.sql` example.

The `DM2` schema has the `SVD_SH_SAMPLE` model and is created by the `oml4sql-singular-value-decomposition.sql`. In the following code, models in `DM1` schema are displayed.

```
SELECT owner, model_name, mining_function, algorithm FROM all_mining_models where
OWNER='DM1';
```

The output is as follows:

```
OWNER      MODEL_NAME           MINING_FUNCTION      ALGORITHM
---------- -------------------- --------------------
-------------------------
DM1        EM_SH_CLUS_SAMPLE    CLUSTERING           EXPECTATION_MAXIMIZATION
DM1        DT_SH_CLAS_SAMPLE    CLASSIFICATION       DECISION_TREE
```

Example 8-1 Creating the Directory Object

```
-- connect as system user
CREATE OR REPLACE DIRECTORY OMLDIR AS '/home/omluser_dir';
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM1;
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM2;
SELECT * FROM all_directories WHERE directory_name = 'OMLDIR';
```

```
OWNER      DIRECTORY_NAME             DIRECTORY_PATH
---------- -------------------------- ----------------------------------------
SYS        OMLDIR                      /home/omluser_dir
```

Example 8-2 Exporting All Models From DM1

```
-- connect as DM1
BEGIN
  dbms_data_mining.export_model (
                  filename =>   'all_DM1',
                  directory =>  'OMLDIR');
END;
/
```

A log file and a dump file are created in `/home/omluser_dir`, the physical directory associated with `OMLDIR`. The name of the log file is `dm1_exp_11.log`. The name of the dump file is `all_dm101.dmp`.

Example 8-3 Importing the Models Back Into DM1

The models that were exported in Example 8-2 still exist in `DM1`. Since an import does not overwrite models with the same name, you must drop the models before importing them back into the same schema.

```
BEGIN
  dbms_data_mining.drop_model('EM_SH_CLUS_SAMPLE');
  dbms_data_mining.drop_model('DT_SH_CLAS_SAMPLE');
  dbms_data_mining.import_model(
                   filename => 'all_dm101.dmp',
                   directory => 'OMLDIR');
END;
/
SELECT model_name FROM user_mining_models;


MODEL_NAME
------------------------------
DT_SH_CLAS_SAMPLE
EM_SH_CLUS_SAMPLE
```

Example 8-4 Importing Models Into a Different Schema

In this example, the models that were exported from `DM1` in Example 8-2 are imported into `DM2`. The `DM1` schema uses the `USER1` tablespace; the `DM2` schema uses the `USER2` tablespace.

```
-- CONNECT as sysdba
BEGIN
  dbms_data_mining.import_model (
                   filename => 'all_d101.dmp',
                   directory => 'OMLDIR',
                   schema_remap => 'DM1:DM2',
                   tablespace_remap => 'USER1:USER2');
END;
/
-- CONNECT as DM2
SELECT model_name from user_mining_models;


MODEL_NAME
--------------------------------------------------------------------------------
--
SVD_SH_SAMPLE
EM_SH_CLUS_SAMPLE
DT_SH_CLAS_SAMPLE
```

Example 8-5 Exporting Specific Models

You can export a single model, a list of models, or a group of models that share certain characteristics.

```
-- Export the model named dt_sh_clas_sample
EXECUTE dbms_data_mining.export_model (
             filename => 'one_model',
```

```
            directory =>'OMLDIR',
            model_filter => 'name in (''DT_SH_CLAS_SAMPLE'')');
-- one_model01.dmp and dm1_exp_37.log are created in /home/omluser_dir

-- Export Decision Tree models
EXECUTE dbms_data_mining.export_model(
            filename => 'algo_models',
            directory => 'OMLDIR',
            model_filter => 'ALGORITHM_NAME IN (''DECISION_TREE'')');
-- algo_model01.dmp and dm1_exp_410.log are created in /home/omluser_dir

-- Export clustering models
EXECUTE dbms_data_mining.export_model(
            filename =>'func_models',
            directory => 'OMLDIR',
            model_filter => 'FUNCTION_NAME = ''CLUSTERING''');
-- func_model01.dmp and dm1_exp_513.log are created in /home/omluser_dir
```

Related Topics

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

8.3.6 EXPORT and IMPORT Serialized Models

From Oracle Database Release 18c onwards, `EXPORT_SERMODEL` and `IMPORT_SERMODEL` procedures are available to export or import serialized models to or from a database.

The serialized format allows the models to be moved to another database instance or OML Services for scoring. The model is exported to a serialized `BLOB` . The import routine takes the serialized content in the `BLOB` and the name of the model to be created with the content.

Related Topics

- EXPORT_SERMODEL Procedure
- IMPORT_SERMODEL Procedure

8.3.7 Import From PMML

You can import regression models represented in Predictive Model Markup Language (PMML).

PMML is an XML-based standard specified by the Data Mining Group (https://www.dmg.org). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Machine Learning for SQL supports the core features of PMML 3.1 for regression models.

You can import regression models represented in PMML. The models must be of type `RegressionModel`, either linear regression or binary logistic regression.

Related Topics

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

8.4 Control Access to Oracle Machine Learning for SQL Models and Data

You can create a Oracle Machine Learning for SQL user and grant necessary privileges by following the steps listed.

- Create an Oracle Machine Learning for SQL User
  An OML4SQL user is a database user account that has privileges for performing machine learning activities.
- System Privileges for Oracle Machine Learning for SQL
  A system privilege confers the right to perform a particular action in the database or to perform an action on a type of schema objects. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.
- Object Privileges for Oracle Machine Learning for SQL Models
  Learn about machine learning object privileges.

8.4.1 Create an Oracle Machine Learning for SQL User

An OML4SQL user is a database user account that has privileges for performing machine learning activities.

Example 8-6 shows how to create a database user. Example 8-7 shows how to assign machine learning privileges to the user.

> **Note**
>
> To create a user for the OML4SQL examples, you must run two configuration scripts as described in Install the OML4SQL Examples.

Example 8-6 Creating a Database User in SQL*Plus

1. Log in to SQL*Plus with system privileges.

```
Enter user-name: sys as sysdba
Enter password: password
```

2. To create a user named `oml_user`, type these commands. Specify a password of your choosing.

```
CREATE USER oml_user IDENTIFIED BY password
       DEFAULT TABLESPACE USERS
       TEMPORARY TABLESPACE TEMP
       QUOTA UNLIMITED ON USERS;
Commit;
```

The `USERS` and `TEMP` tablespaces are included in Oracle AI Database. `USERS` is used mostly by demo users; it is appropriate for running the examples described in About the OML4SQL Examples. `TEMP` is the temporary tablespace that is shared by most database users.

> **Note**
>
> Tablespaces for OML4SQL users must be assigned according to standard DBA practices, depending on system load and system resources.

3. To log in as `oml_user`, enter the following.

```
CONNECT oml_user
Enter password: password
```

- Grant Privileges for Oracle Machine Learning for SQL
  The `CREATE MINING MODEL` is a privilege that you must have to create and perform operations on your model. Some other machine learning privileges can be assigned by issuing `GRANT` statements.

> **See Also**
>
> *Oracle Database SQL Language Reference* for the complete syntax of the `CREATE USER` statement

8.4.1.1 Grant Privileges for Oracle Machine Learning for SQL

The `CREATE MINING MODEL` is a privilege that you must have to create and perform operations on your model. Some other machine learning privileges can be assigned by issuing `GRANT` statements.

You must have the `CREATE MINING MODEL` privilege to create models in your own schema. You can perform any operation on models that you own. This includes applying the model, adding a cost matrix, renaming the model, and dropping the model.

The `GRANT` statements in the following example assign a set of basic machine learning privileges to the `oml_user` account. Some of these privileges are not required for all machine learning activities, however it is prudent to grant them all as a group.

Additional system and object privileges are required for enabling or restricting specific machine learning activities.

The following table lists the system privileges required for running the OML4SQL examples.

Table 8-2 System Privileges Granted by dmshgrants.sql to the OML4SQL User

Privilege	Allows the OML4SQL User To
`CREATE SESSION`	Log in to a database session
`CREATE TABLE`	Create tables, such as the settings tables for `CREATE_MODEL`
`CREATE VIEW`	Create views, such as the views of tables in the `SH` schema
`CREATE MINING MODEL`	Create OML4SQL models
`EXECUTE ON ctxsys.ctx_ddl`	Run procedures in the `ctxsys.ctx_ddl` PL/SQL package; required for text mining

Example 8-7 Privileges Required for Machine Learning

This example grants the required privileges to the user oml_user.

```
GRANT CREATE SESSION TO oml_user;
GRANT CREATE TABLE TO oml_user;
```

```
GRANT CREATE VIEW TO oml_user;
GRANT CREATE MINING MODEL TO oml_user;
GRANT EXECUTE ON CTXSYS.CTX_DDL TO oml_user;
```

READ or SELECT privileges are required for data that is not in your schema. For example, the following statement grants SELECT access to the sh.customers table.

```
GRANT SELECT ON sh.customers TO oml_user;
```

8.4.2 System Privileges for Oracle Machine Learning for SQL

A system privilege confers the right to perform a particular action in the database or to perform an action on a type of schema objects. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

You can perform specific operations on machine learning models in other schemas if you have the appropriate system privileges. For example, CREATE ANY MINING MODEL enables you to create models in other schemas. SELECT ANY MINING MODEL enables you to apply models that reside in other schemas. You can add comments to models if you have the COMMENT ANY MINING MODEL privilege.

To grant a system privilege, you must either have been granted the system privilege with the ADMIN OPTION or have been granted the GRANT ANY PRIVILEGE system privilege.

The system privileges listed in the following table control operations on machine learning models.

Table 8-3 System Privileges for Oracle Machine Learning for SQL

System Privilege	Allows you to....
CREATE MINING MODEL	Create machine learning models in your own schema.
CREATE ANY MINING MODEL	Create machine learning models in any schema.
ALTER ANY MINING MODEL	Change the name or cost matrix of any machine learning model in any schema.
DROP ANY MINING MODEL	Drop any machine learning model in any schema.
SELECT ANY MINING MODEL	Apply a machine learning model in any schema, also view model details in any schema.
COMMENT ANY MINING MODEL	Add a comment to any machine learning model in any schema.
AUDIT_ADMIN role	Generate an audit trail for any machine learning model in any schema. (See *Oracle Database Security Guide* for details.)

Example 8-8 Grant System Privileges for Oracle Machine Learning for SQL

The following statements allow oml_user to score data and view model details in any schema as long as SELECT access has been granted to the data. However, oml_user can only create models in the oml_user schema.

```
GRANT CREATE MINING MODEL TO oml_user;
GRANT SELECT ANY MINING MODEL TO oml_user;
```

The following statement revokes the privilege of scoring or viewing model details in other schemas. When this statement is run, oml_user can only perform machine learning activities in the oml_user schema.

```
REVOKE SELECT ANY MINING MODEL FROM oml_user;
```

Related Topics

- Add a Comment to an Oracle Machine Learning for SQL Model
  You can add a comment to an OML4SQL model object using SQL `COMMENT` statement.
- *Oracle Database Security Guide*

8.4.3 Object Privileges for Oracle Machine Learning for SQL Models

Learn about machine learning object privileges.

An object privilege confers the right to perform a particular action on a specific schema object. For example, the privilege to delete rows from the `SH.PRODUCTS` table is an example of an object privilege.

You automatically have all object privileges for schema objects in your own schema. You can grant object privilege on objects in your own schema to other users or roles.

The object privileges listed in the following table control operations on specific machine learning models.

Table 8-4 Object Privileges for Oracle Machine Learning for SQL Models

Object Privilege	Allows you to....
`ALTER MINING MODEL`	Change the name or cost matrix of the specified machine learning model object.
`SELECT MINING MODEL`	Apply the specified machine learning model object and view its model details.

Example 8-9 Grant Object Privileges on Oracle Machine Learning for SQL Models

The following statements allow `oml_user` to apply the model `testmodel` to the `sales` table, specifying different cost matrixes with each apply. The user `oml_user` can also rename the model `testmodel`. The `testmodel` model and `sales` table are in the `sh` schema, not in the `oml_user` schema.

```
GRANT SELECT ON MINING MODEL sh.testmodel TO oml_user;
GRANT ALTER ON MINING MODEL sh.testmodel TO oml_user;
GRANT SELECT ON sh.sales TO oml_user;
```

The following statement prevents `oml_user` from renaming or changing the cost matrix of `testmodel`. However, `oml_user` can still apply `testmodel` to the `sales` table.

```
REVOKE ALTER ON MINING MODEL sh.testmodel FROM oml_user;
```

8.5 Audit and Add Comments to Oracle Machine Learning for SQL Models

Perform audit of Oracle Machine Learning for SQL model objects through SQL statements.

OML4SQL model objects support SQL `COMMENT` and `AUDIT` statements.

- Add a Comment to an Oracle Machine Learning for SQL Model
  You can add a comment to an OML4SQL model object using SQL `COMMENT` statement.
- Audit Oracle Machine Learning for SQL Models
  Use Oracle AI Database auditing system to audit models to track operations on machine learning models.

8.5.1 Add a Comment to an Oracle Machine Learning for SQL Model

You can add a comment to an OML4SQL model object using SQL `COMMENT` statement.

Comments can be used to associate descriptive information with a database object. You can associate a comment with a machine learning model using a SQL `COMMENT` statement.

```
COMMENT ON MINING MODEL schema_name.model_name IS string;
```

> **Note**
>
> To add a comment to a model in another schema, you must have the `COMMENT ANY MINING MODEL` system privilege.

To drop a comment, set it to the empty `''` string.

The following statement adds a comment to the model `DT_SH_CLAS_SAMPLE` in your own schema.

```
COMMENT ON MINING MODEL dt_sh_clas_sample IS
          'Decision Tree model predicts promotion response';
```

You can view the comment by querying the catalog view `USER_MINING_MODELS`.

```
SELECT model_name, mining_function, algorithm, comments FROM user_mining_models;
```

The output is as follows:

```
MODEL_NAME        MINING_FUNCTION  ALGORITHM      COMMENTS
----------------- ---------------- --------------
-----------------------------------------------
DT_SH_CLAS_SAMPLE CLASSIFICATION   DECISION_TREE  Decision Tree model
predicts promotion response
```

To drop this comment from the database, issue the following statement:

```
COMMENT ON MINING MODEL dt_sh_clas_sample '';
```

> **See Also**
>
> - Table 8-3
> - *Oracle Database SQL Language Reference* for details about SQL `COMMENT` statements

8.5.2 Audit Oracle Machine Learning for SQL Models

Use Oracle AI Database auditing system to audit models to track operations on machine learning models.

The Oracle AI Database auditing system is a powerful, highly configurable tool for tracking operations on schema objects in a production environment. The auditing system can be used to track operations on machine learning models.

> **Note**
>
> To audit machine learning models, you must have the `AUDIT_ADMIN` role.

Unified auditing is documented in *Oracle Database Security Guide*. However, the full unified auditing system is not enabled by default. Instructions for migrating to unified auditing are provided in *Oracle Database Upgrade Guide*.

> **See Also**
>
> - "Auditing Oracle Machine Learning for SQL Events" in *Oracle Database Security Guide* for details about auditing machine learning models
> - "Monitoring Database Activity with Auditing" in *Oracle Database Security Guide* for a comprehensive discussion of unified auditing in Oracle AI Database
> - "About the Unified Auditing Migration Process for Oracle AI Database" in *Oracle Database Upgrade Guide* for information about migrating to unified auditing
> - *Oracle Database Upgrade Guide*

A

Oracle Machine Learning for SQL Examples

Describes the OML4SQL examples.

- About the OML4SQL Examples
  The OML4SQL examples illustrate typical approaches to data preparation, algorithm selection, algorithm tuning, testing, and scoring.
- Install the OML4SQL Examples
  Learn how to install OML4SQL examples.
- OML4SQL Sample Data
  The data used by the OML4SQL examples is based on these tables in the `SH` schema.

A.1 About the OML4SQL Examples

The OML4SQL examples illustrate typical approaches to data preparation, algorithm selection, algorithm tuning, testing, and scoring.

You can learn a great deal about the OML4SQL application programming interface from the OML4SQL examples. The examples are simple. They include extensive inline comments to help you understand the code. They delete all temporary objects on exit so that you can run the examples repeatedly without setup or cleanup.

The OML4SQL examples are available on GitHub at https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql/. Select the Database release (for example 26ai) to see the examples.

The OML4SQL examples create a set of machine learning models in the user's schema. The following table lists the file name of the example and the `mining_function` value and algorithm the example uses.

Table A-1 Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
`oml4sql-anomaly-detection-1class-svm.sql`	`CLASSIFICATION`	`ALGO_SUPPORT_VECTOR_MACHINE`
`oml4sql-anomaly-detection-em.sql`	`CLASSIFICATION`	`ALGO_EXPECTATION_MAXIMIZATION`
`oml4sql-association-rules.sql`	`ASSOCIATION`	`ALGO_APRIORI_ASSOCIATION_RULES`
`oml4sql-classification-decision-tree.sql`	`CLASSIFICATION`	`ALGO_DECISION_TREE`
`oml4sql-classification-glm.sql`	`CLASSIFICATION`	`ALGO_GENERALIZED_LINEAR_MODEL`
`oml4sql-classification-naive-bayes.sql`	`CLASSIFICATION`	`ALGO_NAIVE_BAYES`
`oml4sql-classification-neural-networks.sql`	`CLASSIFICATION`	`ALGO_NEURAL_NETWORK`
`oml4sql-classification-random-forest.sql`	`CLASSIFICATION`	`ALGO_RANDOM_FOREST`

Table A-1 (Cont.) Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
oml4sql-classification-regression-xgboost.sql	CLASSIFICATION	ALGO_XGBOOST
oml4sql-classification-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-classification-text-analysis-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-clustering-expectation-maximization.sql	CLUSTERING	ALGO_EXPECTATION_MAXIMIZATION
oml4sql-clustering-kmeanms-star-schema.sql	CLUSTERING	ALGO_KMEANS
oml4sql-clustering-kmeans.sql	CLUSTERING	ALGO_KMEANS
oml4sql-clustering-ocluster.sql	CLUSTERING	ALGO_O_CLUSTER
oml4sql-cross-validation-decision-tree.sql	CLASSIFICATION	ALGO_DECISION_TREE
oml4sql-feature-extraction-cur.sql	ATTRIBUTE_IMPORTANCE	ALGO_CUR_DECOMPOSITION
oml4sql-feature-extraction-nmf.sql	FEATURE_EXTRACTION	ALGO_NONNEGATIVE_MATRIX_FACTOR
oml4sql-feature-extraction-svd.sql	FEATURE_EXTRACTION	ALGO_SINGULAR_VALUE_DECOMP
oml4sql-feature-extraction-text-mining-esa.sql	FEATURE_EXTRACTION	ALGO_EXPLICIT_SEMANTIC_ANALYS
oml4sql-feature-extraction-text-mining-nmf.sql	FEATURE_EXTRACTION	ALGO_NONNEGATIVE_MATRIX_FACTOR
oml4sql-feature-extraction-text-term-extraction.sql	FEATURE_EXTRACTION	ALGO_EXPLICIT_SEMANTIC_ANALYSIS
oml4sql-partitioned-models-svm.sql	CLASSIFICATION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-regression-glm.sql	REGRESSION	ALGO_GENERALIZED_LINEAR_MODEL
oml4sql-regression-neural-networks.sql	REGRESSION	ALGO_NEURAL_NETWORK
oml4sql-regression-random-forest.sql	REGRESSION	ALGO_RANDOM_FOREST
oml4sql-regression-svm.sql	REGRESSION	ALGO_SUPPORT_VECTOR_MACHINES
oml4sql-singular-value-decomposition.sql	REGRESSION	ALGO_SINGULAR_VALUE_DECOMPOSITION
oml4sql-survival-analysis-xgboost.sql	REGRESSION	ALGO_XGBOOST
oml4sql-time-series-esm-auto-model-search.sql	TIME_SERIES	ALGO_EXPONENTIAL_SMOOTHING
oml4sql-time-series-exponential-smoothing.sql	TIME_SERIES	ALGO_EXPONENTIAL_SMOOTHING
oml4sql-time-series-mset.sql	CLASSIFICATION	ALGO_MSET_SPRT
oml4sql-time-series-regression-dataset.sql	-	This is a dataset to construct time series regression model.

Table A-1 (Cont.) Models Created by Examples

File Name	MINING_FUNCTION	Algorithm
oml4sql-time-series-regression.sql	TIME_SERIES and REGRESSION	Uses ALGO_EXPONENTIAL_SMOOTHING, ALGO_GENERALIZED_MODEL, and ALGO_XGBOOST

A few examples other than those listed in the table above are: `oml4sql-attribute-importance.sql`, which uses the `DBMS_PREDICTIVE_ANALYTICS.EXPLAIN` procedure to find the importance of attributes that independently impact the target attribute. `oml4sql-feature-extraction-text-term-extraction.sql` example, which uses the `CTX.DDL` package for text extraction.

Another set of examples demonstrates the use of the `ALGO_EXTENSIBLE_LANG` algorithm to register R language functions and create R models. The following table lists the R Extensibility examples. It shows the file name of the example and the MINING_FUNCTION value and R function used.

File Name	MINING_FUNCTION	R Function
oml4sql-r-extensible-algorithm-registration.sql	CLASSIFICATION	glm
oml4sql-r-extensible-association-rules.sql	ASSOCIATION	apriori
oml4sql-r-extensible-attribute-importance-via-rf.sql	REGRESSION	randomForest
oml4sql-r-extensible-glm.sql	REGRESSION	glm
oml4sql-r-extensible-kmeans.sql	CLUSTERING	kmeans
oml4sql-r-extensible-principal-components.sql	FEATURE_EXTRACTION	prcomp
oml4sql-r-extensible-regression-tree.sql	REGRESSION	rpart
oml4sql-r-extensible-regression-neural-networks.sql	REGRESSION	nnet

A.2 Install the OML4SQL Examples

Learn how to install OML4SQL examples.

The OML4SQL examples require:

- Oracle AI Database (on-premises, Oracle AI Database Cloud Service, or Oracle Autonomous AI Database)
- Oracle AI Database sample schemas
- A user account with the privileges described in Grant Privileges for Oracle Machine Learning for SQL.
- Running of `dmshgrants.sql` by a system administrator
- Running of `dmsh.sql` by the OML4SQL user

Follow these steps to install the OML4SQL examples:

1. Install or obtain access to an Oracle AI Database 26ai instance. To install the database, see the installation instructions for your platform at Oracle AI Database 26ai.
2. Ensure that the sample schemas are installed in the database. See *Oracle Database Sample Schemas* for details about the sample schemas.
3. Download the example code files from GitHub at https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql. Select the Database edition. Place the files in a directory to which you have access on the Oracle AI Database server. For example, `$ORACLE_HOME/demo/schema`. `$ORACLE_HOME` is the home path where you have installed the database. Typically, `/u01/app/oracle/product/23.0.0/dbhome_1`.
4. Verify that your user account has the required privileges described in Grant Privileges for Oracle Machine Learning for SQL.
5. Ask your system administrator to run the `dmshgrants.sql` script, or run it yourself if you have administrative privileges. The script grants the privileges that are required for running the examples. These include `SELECT` access to tables in the `SH` schema as described in OML4SQL Sample Data and the system privileges.

   Connect as `SYSDBA`:

   ```
   CONNECT sys / as sysdba
   Enter password: sys_password
   Connected.
   ```

   Pass the name of the OML4SQL user to `dmshgrants`:

   ```
   @<location_of_examples>/dmshgrants oml_user
   ```

6. Connect to the database and run the `dmsh.sql` script. This script creates views of the sample data in the schema of the OML4SQL user.

   ```
   CONNECT oml_user
   Enter password: oml_user_password
   Connected.
   ```

   Issue the following to run the script:

   ```
   @<location_of_examples>/dmsh.sql
   ```

Related Topics

- *Oracle Database Sample Schemas*

A.3 OML4SQL Sample Data

The data used by the OML4SQL examples is based on these tables in the `SH` schema.

Those tables are:

```
SH.CUSTOMERS
SH.SALES
SH.PRODUCTS
SH.SUPPLEMENTARY_DEMOGRAPHICS
SH.COUNTRIES
```

The `dmshgrants` script grants `SELECT` access to the tables in the `SH` schema. The `dmsh.sql` script creates views of the `SH` tables in the schema of the OML4SQL user. The views are described in the following table.

Table A-2 Views Created by dmsh.sql

View Name	Description
`MINING_DATA`	Joins and filters data
`MINING_DATA_BUILD_V`	Data for building models
`MINING_DATA_TEST_V`	Data for testing models
`MINING_DATA_APPLY_V`	Data to be scored
`MINING_BUILD_TEXT`	Data for building models that include text
`MINING_TEST_TEXT`	Data for testing models that include text
`MINING_APPLY_TEXT`	Data, including text columns, to be scored
`MINING_DATA_ONE_CLASS_V`	Data for anomaly detection

The association rules example creates its own transactional data.

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X