# Oracle® Data Mining

## User's Guide

19c
E97868-04
November 2023

Oracle Data Mining User's Guide, 19c

E97868-04



Primary Author: Sarika Surampudi

# Contents

## 3 Preparing the Data



## 4 Transforming the Data

## 5 Creating a Model

## 6 Scoring and Deployment

# List of Tables

# Preface

This guide explains how to use the programmatic interfaces to Oracle Data Mining and how to use features of Oracle Database to administer Oracle Data Mining. This guide presents the tools and procedures for implementing the concepts that are presented in *Oracle Data Mining Concepts*.

This preface contains these topics:



## Audience

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

## Documentation Accessibility

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

**Access to Oracle Support**

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

## Diversity and Inclusion

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

# Related Documentation

Oracle Data Mining components associated with Oracle Database are included with the database license. The Oracle Data Mining for SQL documents are available from the Data Warehousing and Business Intelligence page of the Oracle Database online documentation library:

`Oracle Database Data Warehousing`

The following manuals document Oracle Data Mining:

- *Oracle Data Mining Concepts*
- *Oracle Data Mining User's Guide* (this guide)
- *Oracle Data Mining API Guide*

> **Note:**
>
> The virtual book combines key passages from the two Data Mining 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* (Data Mining 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 Data Mining Resources on the Oracle Technology Network

The Oracle Data Mining page on the Oracle Technology Network (OTN) provides a wealth of information, including white papers, demonstrations, blogs, discussion forums, and Oracle By Example tutorials:

`Oracle Data Mining`

You can download Oracle Data Miner, the graphical user interface to Oracle Data Mining, 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 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 Data Mining User's Guide

Changes in this release for *Oracle Data Mining User's Guide*.

## Oracle Data Mining User's Guide

- This guide was introduced in release 12*c*. *Oracle Data Mining User's Guide* replaces two manuals that were provided in previous releases: *Oracle Data Mining Administrator's Guide* and *Oracle Data Mining Application Developer's Guide*.
- Information about database administration for Oracle Data Mining is now consolidated in Administrative Tasks for Oracle Data Mining . The remaining chapters of this guide are devoted to application development.

## Changes in Oracle Data Mining 19c

The following changes are documented in *Oracle Data Mining User's Guide* for 19c.

### Deprecated Features

The following features are deprecated in this release, and may be desupported in another release. See *Oracle Database Upgrade Guide* for a complete list of deprecated features in this release.

- `*GET_MODEL_DETAILS` are deprecated and are replaced with *Model Detail Views*. See Model Detail Views.

### Desupported Features

See *Oracle Database Upgrade Guide* for a complete list of desupported features in this release.

### Other Changes

The following is an additional change in *Oracle Data Mining User's Guide* for 19c:

- "Enabling or Disabling a Database Option" topic is removed from the publication as the information is obsolete.
- Exporting and Importing Mining Models chapter is updated.
- Moved "Outlier Treatment" topic under "Oracle Data Mining Transformation Routines" topic.

1

# Data Mining With SQL

Learn how to solve business problems using the Oracle Data Mining application programming interface (API).



## 1.1 Highlights of the Data Mining API

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

Data mining 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. Data mining also has critical applications in the public sector and in scientific research. However, the complexities of data mining application development and the complexities inherent in managing and securing large stores of data can limit the adoption of data mining technology.

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

The Oracle Data Mining 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 data mining 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:**
>
> A set of sample data mining programs ship with Oracle Database. The examples in this manual are taken from these samples.

**Related Topics**

- The Data Mining Sample Programs
  Describes the data mining sample programs that ship with Oracle Database.
- *Oracle Data Mining Concepts*

# 1.2 Example: Targeting Likely Candidates for a Sales Promotion

This example targets 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.

**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';

  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';

  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;

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.

```
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;

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;
```

```
  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 executed 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;

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 data mining operations without a predefined model. In this example, the `EXPLAIN` routine computes the top predictors for affinity card ownership. 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;

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

```
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;

      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;

   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;

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 : Building an ESA Model with a Wiki Dataset

The examples shows `FEATURE_COMPARE` function with Explicit Semantic Analysis (ESA) model, which compares a similar set of texts and then a dissimilar set of texts.

The example shows an ESA model built against a 2005 Wiki dataset rendering over 200,000 features. The documents are mined as text and the document titles are given as the feature IDs.

**Similar Texts**

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

SIMILARITY
----------
      .258
```

The output metric shows 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;

SIMILARITY
----------
      .007
```

2

# About the Data Mining API

Overview of the Oracle Data Mining application programming interface (API) components.

- About Mining Models
- Data Mining Data Dictionary Views
- Data Mining PL/SQL Packages
- Data Mining SQL Scoring Functions

13

## 2.1 About Oracle Machine Learning Models

Data mining models are database schema objects that perform data mining techniques.

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

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

13

Along with the data mining technique, in the `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 data mining technique, and some are specific to an algorithm.

> **Note:**
>
> Most types of data mining 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**

- Dynamic Scoring
- DBMS_PREDICTIVE_ANALYTICS
  Understand the routines of `DBMS_PREDICTIVE_ANALYTICS` package.
- Creating a Model
  Explains how to create data mining models and query model details.
- Administrative Tasks for Oracle Data Mining
  Explains how to perform administrative tasks related to Oracle Data Mining.

# 2.2 Data Mining Data Dictionary Views

Lists Oracle Data Mining data dictionary views.

The data dictionary views for Oracle Data Mining 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 Data Mining**

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

## 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
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
```

**Related Topics**

- Creating a Model
  Explains how to create data mining models and query model details.
- *Oracle Database Reference*

## 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
 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.

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

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**

- About the Data Mining API
  Overview of the Oracle Data Mining application programming interface (API) components.
- *Oracle Database Reference*

## 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
 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.

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

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**

- *Oracle Database Reference*

## 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 mining function, or they may be general.

**Example 2-4 ALL_MINING_MODEL_SETTINGS**

```
describe ALL_MINING_MODEL_SETTINGS
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.

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

SETTING_NAME                   SETTING_VALUE                  SETTING
------------------------------ ------------------------------ -------
ALGO_NAME                      ALGO_SINGULAR_VALUE_DECOMP     INPUT
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**

- Specifying Model Settings
  Understand how to configure data mining models at build time.
- *Oracle Database Reference*

## 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
 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 a model `SVD_SH_SAMPLE`. The model uses the Singular Value Decomposition algorithm for feature extraction.

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

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**

- *Oracle Database Reference*

## 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`.

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

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**

- *Oracle Database Reference*

# 2.3 Data Mining PL/SQL Packages

The PL/SQL interface to Oracle Data Mining is implemented in three packages.

The following table displays the PL/SQL packages.

**Table 2-2 Data Mining PL/SQL Packages**

| Package Name | Description |
|---|---|
| DBMS_DATA_MINING | Routines for creating and managing mining models |
| DBMS_DATA_MINING_TRANSFORM | Routines for transforming the data for mining |
| DBMS_PREDICTIVE_ANALYTICS | Routines that perform predictive analytics |

**Related Topics**

- DBMS_DATA_MINING
- DBMS_DATA_MINING_TRANSFORM
- DBMS_PREDICTIVE_ANALYTICS

## 2.3.1 DBMS_DATA_MINING

Understand the routines of DBMS_DATA_MINING package.

The DBMS_DATA_MINING package contains routines for creating mining models, for performing operations on mining models, and for querying mining models. The package includes routines for:

- Creating, dropping, and performing other DDL operations on mining 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's 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

Understand the routines of DBMS_DATA_MINING_TRANSFORM package.

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 mining model.
- Specifying transformations as relational views (external to mining model objects).
- Specifying distinct properties for columns in the build data. For example, you can specify that the column must be interpreted as unstructured text, or that the column must be excluded from Automatic Data Preparation.

**Related Topics**

- Transforming the Data
  Understand how to transform data for building a model or for scoring.
- *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 |
| `SET_TRANSFORM` | Specifies transformations for embedding in a model |

The statements in the following example create an 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.

**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

Understand the routines of `DBMS_PREDICTIVE_ANALYTICS` package.

The `DBMS_PREDICTIVE_ANALYTICS` package contains routines that perform an automated form of data mining known as predictive analytics. With predictive analytics, you do not need to be aware of model building or scoring. All mining 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 Data Mining SQL Scoring Functions

Understand the different data mining SQL scoring functions.

The Data Mining SQL language functions use Oracle Data Mining to score data. The functions can apply a mining model schema object to the data, or they can dynamically mine the data by executing an analytic clause. SQL functions are available for all the data mining algorithms that support the scoring operation. All Data Mining SQL functions, as listed in the following table can operate on R Mining Model with the corresponding mining function. However, the functions are not limited to the ones listed here.

**Table 2-4 Data Mining SQL 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. |

**Table 2-4 (Cont.) Data Mining SQL Functions**

| Function | Description |
|---|---|
| PREDICTION_COST | Returns a measure of the cost of incorrect predictions |
| PREDICTION_DETAILS | Returns detailed information about the prediction |
| PREDICTION_PROBABILITY | Returns the probability of the prediction |
| PREDICTION_SET | Returns the results of a classification model, including the predictions and associated probabilities for each case |

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.

### 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;


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

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

### Related Topics

- Scoring and Deployment
  Explains the scoring and deployment features of Oracle Data Mining.
- *Oracle Database SQL Language Reference*

# 3
# Preparing the Data

Learn how to create a table or view that can be used to build a model.



## 3.1 Data Requirements

Understand how data is stored and viewed for data mining.

Data mining 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 mining. 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 Data Mining 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**

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

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

**Related Topics**

- Using Market Basket Data

## 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.

Oracle Data Mining supports standard Oracle data types as well as the following collection types:

```
DM_NESTED_CATEGORICALS
DM_NESTED_NUMERICALS
DM_NESTED_BINARY_DOUBLES
DM_NESTED_BINARY_FLOATS
```

**Related Topics**

- Using Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- Mining Unstructured Text
  Explains how to use Oracle Data Mining to mine unstructured text.
- *Oracle Database SQL Language Reference*

## 3.1.2 Data Sets for Classification and Regression

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

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

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

## 3.1.3 Scoring Requirements

Most data mining models can be applied to separate data in a process known as **scoring**. Oracle Data Mining 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, Oracle Data Mining 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, Oracle Data Mining 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:**
>
> Oracle Data Mining 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.
>
> Oracle Data Mining 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:**
>
> Transforming the Data for more information on automatic and embedded data transformations

# 3.2 About Attributes

Attributes are the items of data that are used in data mining. 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.

## 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**

- Using Nested Data
  A join between the tables for one-to-many relationship is represented through nested columns.
- Transforming the Data
  Understand how to transform data for building a model or for scoring.

> **See Also:**

## 3.2.2 Target Attribute

Understand what a **target** means in data mining 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**

| Mining Function | Target Data Types |
|---|---|
| Classification | `VARCHAR2`, `CHAR` |
| | `NUMBER`, `FLOAT` |
| | `BINARY_DOUBLE`, `BINARY_FLOAT`, `ORA_MINING_VARCHAR2_NT` |
| 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 Data Mining interprets `NUMBER`, `FLOAT`, `BINARY_DOUBLE`, `BINARY_FLOAT`, `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.

Oracle Data Mining interprets `CHAR` and `VARCHAR2` as categorical by default, however these columns may also be identified as columns of unstructured data (text). Oracle Data Mining 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.
- Mining Unstructured Text
  Explains how to use Oracle Data Mining to mine unstructured text.

## 3.2.4 Model Signature

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

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
  The GET_* interfaces are replaced by model views, and Oracle recommends that users leverage the views instead.

# 3.3 Using Nested Data

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

Oracle Data Mining 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.

Oracle Data Mining 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 Data Mining nested table types. Each row in the nested column consists of an attribute name/value pair. Oracle Data Mining internally processes each nested row as a separate attribute.

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

**Related Topics**

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

## 3.3.1 Nested Object Types

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

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

Oracle Data Mining 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 Oracle Data Mining 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 Mining

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 mining 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 mining. 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 mining at the product case level, because the information for each case is stored in a single row.

Oracle Data Mining 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 Using Market Basket Data

Market basket data identifies the items sold in a set of baskets or transactions. Oracle Data Mining provides the association mining 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. Oracle Data Mining 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, Oracle Data Mining requires that the data be presented in single-record case format.

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

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

## 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 mining.

```
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;

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

**Related Topics**

- Handling Missing Values

# 3.5 Using Retail Analysis Data

Retail analysis often makes use of Association Rules and Association models.

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

**Related Topics**

- *Oracle Data Mining Concepts*

## 3.5.1 Example: Calculating Aggregates

The following example shows the concept of Aggregates.

**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 |

**Table 3-2 (Cont.) Grocery Store Data**

| Customer | Item A | Item B | Item C | Item D |
|---|---|---|---|---|
| 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 Handling Missing Values

Oracle Data Mining 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).

Oracle Data Mining 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.

## 3.6.1 Examples: Missing Values or Sparse Data?

The examples in this section illustrate how Oracle Data Mining identifies data as either sparse or missing at random.

#### 3.6.1.1 Sparsity in a Sales Table

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 mining.

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 Data Mining 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

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 simply 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 Data Mining

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:**
>
> Oracle Data Mining performs the same missing value treatment whether or not Automatic Data Preparation is being used.

**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.<br>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 | Genelized Linear Models (GLM), Non-Negative Matrix Factorization (NMF), *k*-Means, and Support Vector Machine (SVM) replaces missing categorical values with the mode.<br>Singular Value Decomposition (SVD) does not support categorical data.<br>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) and 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 Data Mining 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 Oracle Data Mining interprets them as missing at random, you can use a SQL function like `NVL` to replace the nulls with a value such as "NA". Oracle Data Mining 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 1000 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*

# 4
# Transforming the Data

Understand how to transform data for building a model or for scoring.



## 4.1 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 data mining 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 Data Mining supports Automatic Data Preparation (ADP), which automatically implements the transformations required by the algorithm. The transformations are embedded in the model and automatically executed 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 just 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 just 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

# 4.2 Preparing the Case Table

Understand why you have to prepare a case table.

The first step in preparing data for mining 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.

**Related Topics**

- Preparing the Data
  Learn how to create a table or view that can be used to build a model.

## 4.2.1 Creating Nested Columns

Learn when to create nested columns.

When the data source includes transactional data (multi-record case), the transactions must be aggregated to the case level in nested columns. In transactional data, the information for each case is contained in multiple rows. An example is sales data in a star schema when mining at the product level. Sales is stored in many rows for a single product (the case) since 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

## 4.2.2 Converting Column Data Types

You must convert the data type of a column if its type causes Oracle Data Mining to interpret it incorrectly. 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')
```

## 4.2.3 Text Transformation

You can use Oracle Data Mining to mine text. Columns of text in the case table can be mined 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`.

### 4.2.4 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 abnormalities. In others, they provide meaningful information.

**Related Topics**

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

## 4.3 Understanding Automatic Data Preparation

Understand data transformation using Automatic Data Preparation (ADP).

Most algorithms require some form of data transformation. During the model build process, Oracle Data Mining 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, Oracle Data Mining 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 data mining algorithms.

**Related Topics**

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

### 4.3.1 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.

## 4.3.2 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).

## 4.3.3 How ADP Transforms the Data

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

**Table 4-1 Oracle Data Mining Algorithms With ADP**

| Algorithm | Mining Function | Treatment by ADP |
|---|---|---|
| Apriori | Association Rules | ADP has no effect on association rules. |
| 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. |
| Naive Bayes | Classification | All attributes are binned with supervised binning. |
| 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. |
| SVD | Feature Extraction | Numerical attributes are normalized. |
| SVM | Classification, Anomaly Detection, and Regression | Numerical attributes are normalized. |

**See Also:**

- *Oracle Database PL/SQL Packages and Types Reference*
- Part III of *Oracle Data Mining Concepts* for more information about algorithm-specific data preparation

# 4.4 Embedding 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_MODEL`.

```
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);
```

## 4.4.1 Specifying Transformation Instructions for an Attribute

Learn what is a transformation instruction for an attribute and learn about the fields in a transformation record.

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 4-2 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<br>`CASE WHEN age < 19 THEN 'child' ELSE 'adult'`<br>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:<br>`DECODE(age,'child','(-Inf,19)','[19,Inf)')` |

**Table 4-2 (Cont.) Fields in a Transformation Record for an Attribute**

| Field | Description |
|---|---|
| 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:<br>• 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.<br>• 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.<br>• 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.<br>See Example 4-1 and Example 4-2. |

**Related Topics**

- Scoping of Model Attribute Name
- Expression Records

## 4.4.1.1 Expression Records

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.

## 4.4.1.2 Attribute Specifications

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

The attribute specification in a transformation record defines characteristics that are specific to this attribute. If not null, the attribute specification can include values FORCE_IN, NOPREP, or TEXT, as described in Table 4-2.

**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**

- Configuring a Text Attribute
  Learn how to identify a column as a text attribute and provide transformation instructions for any text attribute.

## 4.4.2 Building a Transformation List

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`

### 4.4.2.1 SET_TRANSFORM

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.

### 4.4.2.2 The STACK Interface

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

The STACK interface 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.

### 4.4.2.3 GET_MODEL_TRANSFORMATIONS and GET_TRANSFORM_LIST

Use the functions to create a new transformation list.

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

The GET_MODEL_TRANSFORMATIONS function returns a list of embedded transformations.

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

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

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

The components of a transformation list are transform_rec, not dm_transform. The fields of a transform_rec are described in Table 4-2. 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*

## 4.4.3 Transformation Lists and Automatic Data Preparation

If you enable ADP and you specify a transformation list, the transformation list is embedded with the automatic, system-generated transformations. The transformation list is executed before the automatic transformations.

If you enable ADP and do not specify a transformation list, only the automatic transformations are embedded in the model.

If ADP is disabled (the default) and you specify a transformation list, your custom transformations are embedded in the model. No automatic transformations are performed.

If ADP is disabled (the default) and you do not specify a transformation list, no transformations is embedded in the model. You have to transform the training, test, and scoring data sets yourself if necessary. You must take care to apply the same transformations to each data set.

## 4.4.4 Oracle Data Mining Transformation Routines

Learn about transformation routines.

Oracle Data Mining provides routines that implement various transformation techniques in the `DBMS_DATA_MINING_TRANSFORM` package.

**Related Topics**

- *Oracle Database SQL Language Reference*

### 4.4.4.1 Binning Routines

Explains Binning techniques in Oracle Data Mining.

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 Oracle Data Mining:

**Table 4-3 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

## 4.4.4.2 Normalization Routines

Learn about Normalization routines in Oracle Data Mining.

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 4-4 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

### 4.4.4.3 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.

### 4.4.4.4 Routines 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 4-5 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 Understanding Reverse Transformations

Understand why you need 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 donot 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, Oracle Data Mining explodes nested columns, and each row (an attribute name/value pair) becomes an attribute.

Some algorithms, for example Support Vector Machines (SVM) and Generalized Linear Models (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 regards to interpretability of 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 donot 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
  The `GET_*` interfaces are replaced by model views, and Oracle recommends that users leverage the views instead.

# 5
# Creating a Model

Explains how to create data mining models and query model details.



## 5.1 Before Creating a Model

Explains the preparation steps before creating a model.

Models are database schema objects that perform data mining. The `DBMS_DATA_MINING` PL/SQL package is the API for creating, configuring, evaluating, and querying mining 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 5-1 Preparation for Creating a Mining Model**

| Preparation Step | Description |
|---|---|
| Choose the mining function | See "Choosing the Mining Function" |
| Choose the algorithm | See "Choosing the Algorithm" |
| Identify the build (training) data | See "Preparing the Data" |
| For classification models, identify the test data | See "Data Sets for Classification and Regression" |
| Determine your data transformation strategy | See " Transforming the Data" |
| Create and populate a settings tables (if needed) | See "Specifying Model Settings" |

**Related Topics**

- About Oracle Machine Learning Models
  Data mining models are database schema objects that perform data mining techniques.
- DBMS_DATA_MINING
  Understand the routines of `DBMS_DATA_MINING` package.

## 5.2 The CREATE_MODEL Procedure

The `CREATE_MODEL` procedure in the `DBMS_DATA_MINING` package uses the specified data to create a mining model with the specified name and mining 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);
```

## 5.2.1 Choosing the Mining Technique

Explains about providing mining technique to `CREATE_MODEL`.

The mining technique is a required argument to the `CREATE_MODEL` procedure. A data mining technique specifies a class of problems that can be modeled and solved.

Data mining techniques 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 techniques are predictive. Unsupervised techniques are descriptive.

> **Note:**
>
> In data mining terminology, a **technique** is a general type of problem to be solved by a given approach to data mining. In SQL language terminology, a **function** is an operator that returns a value.
>
> In Oracle Data Mining documentation, the term **technique**, or **mining technique** refers to a data mining technique; the term **SQL function** or **SQL Data Mining function** refers to a SQL function for scoring (applying data mining models).

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

**Table 5-2 Mining Model Techniques**

| ***Mining_Function*** **Value** | **Description** |
|---|---|
| `ASSOCIATION` | Association is a descriptive mining technique. An association model identifies relationships and the probability of their occurrence within a data set. (association rules)<br>Association models use the Apriori algorithm. |
| `ATTRIBUTE_IMPORTANCE` | Attribute Importance is a predictive mining technique. An attribute importance model identifies the relative importance of attributes in predicting a given outcome.<br>Attribute Importance models use the Minimum Description Length algorithm and CUR Matrix Decomposition. |

**Table 5-2 (Cont.) Mining Model Techniques**

| *Mining_Function* Value | Description |
|---|---|
| CLASSIFICATION | Classification is a predictive mining technique. A classification model uses historical data to predict a categorical target.<br>Classification models can use Naive Bayes, Neural Network, Decision Tree, Logistic Regression, Random Forest, Support Vector Machines, or Explicit Semantic Analysis. The default is Naive Bayes.<br>The classification technique can also be used for anomaly detection. In this case, the SVM algorithm with a null target is used (One-Class SVM). |
| CLUSTERING | Clustering is a descriptive mining technique. A clustering model identifies natural groupings within a data set.<br>Clustering models can use k-Means, O-Cluster, or Expectation Maximization. The default is *k*-Means. |
| FEATURE_EXTRACTION | Feature Extraction is a descriptive mining technique. A feature extraction model creates a set of optimized attributes.<br>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 mining technique. A regression model uses historical data to predict a numerical target.<br>Regression models can use Support Vector Machines or Linear Regression. The default is Support Vector Machine. |
| TIME_SERIES | Time series is a predictive mining technique. 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**

- *Oracle Data Mining Concepts*

## 5.2.2 Choosing 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 mining technique, or if there is only one algorithm available for the mining technique, you do not need to specify the ALGO_NAME setting. Instructions for specifying model settings are in "Specifying Model Settings".

**Table 5-3 Data Mining Algorithms**

| ALGO_NAME Value | Algorithm | Default? | Mining Model Technique |
|---|---|---|---|
| ALGO_AI_MDL | Minimum Description Length | — | attribute importance |
| ALGO_APRIORI_ASSOCIATION_RULES | Apriori | — | association |

**Table 5-3 (Cont.) Data Mining Algorithms**

| ALGO_NAME Value | Algorithm | Default? | Mining Model Technique |
|---|---|---|---|
| ALGO_CUR_DECOMPOSITION | CUR Decomposition | | Attribute Importance |
| ALGO_DECISION_TREE | Decision Tree | — | classification |
| ALGO_EXPECTATION_MAXIMIZATION | Expectation Maximization | | |
| ALGO_EXPLICIT_SEMANTIC_ANALYS | Explicit Semantic Analysis | — | feature extraction<br>classification |
| ALGO_EXPONENTIAL_SMOOTHING | Exponential Smoothing | — | time series |
| ALGO_EXTENSIBLE_LANG | Language used for extensible algorithm | — | All mining techniques are supported |
| ALGO_GENERALIZED_LINEAR_MODEL | Generalized Linear Model | — | classification and regression |
| ALGO_KMEANS | *k*-Means | yes | clustering |
| 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<br>regression, classification, and anomaly detection (classification with no target) |

**Related Topics**

- Specifying Model Settings
  Understand how to configure data mining models at build time.
- *Oracle Data Mining Concepts*

## 5.2.3 Supplying Transformations

You can optionally specify transformations for the build data in the *xform_list* parameter to CREATE_MODEL. The transformation instructions are embedded in the model and reapplied whenever the model is applied to new data.

### 5.2.3.1 Creating a Transformation List

The following are the ways to create a transformation list:

- The STACK interface in DBMS_DATA_MINING_TRANSFORM.

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

- The SET_TRANSFORM procedure in DBMS_DATA_MINING_TRANSFORM.

  The SET_TRANSFORM procedure applies a specified SQL expression to a specified attribute. For example, the following statement appends a transformation instruction for country_id to a list of transformations called my_xforms. The transformation instruction divides country_id by 10 before algorithmic processing begins. The reverse transformation multiplies country_id by 10.

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

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

### 5.2.3.2 Transformation List and Automatic Data Preparation

Understand the interaction between transformation list and Automatic Data Preparation (ADP).

The transformation list argument to 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 executed 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**

- Embedding Transformations in a Model
- *Oracle Database PL/SQL Packages and Types Reference*

## 5.2.4 About Partitioned Model

Introduces partitioned model to organise and represent multiple models.

Oracle Data Mining supports building of a persistent Oracle Data Mining partitioned model. A partitioned model organizes and represents multiple models as partitions in a single model entity, enabling a user to easily build and manage models tailored to independent slices of data. Persistent means that the partitioned model has an on-disk representation. The product manages the organization of the partitioned model and simplifies the process of scoring the partitioned model. You must include the partition columns as part of the USING clause when scoring.

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

**Related Topics**

- *Oracle Database Reference*
- *Oracle Data Mining User's Guide*

## 5.2.4.1 Partitioned Model Build Process

To build a Partitioned Model, Oracle Data Mining requires a partitioning key. The partition key is set through a build setting in the 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. This 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 exceed the defined maximum, Oracle Data Mining 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 mining function
- The mining algorithm
- A super set of all mining 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

## 5.2.4.2 DDL in Partitioned model

Partitioned models are maintained through the following DDL operations:

- Drop model or drop partition
- Add partition

### 5.2.4.2.1 Drop Model or Drop Partition

Oracle Data Mining supports dropping a single model partition for a given partition name.

If only a single partition remains, you cannot explicitly drop that partition. Instead, you must either add additional partitions prior to dropping the partition or you may choose to drop the model itself. When dropping a partitioned model, all partitions are dropped in a single atomic operation. From a performance perspective, Oracle recommends `DROP_PARTITION` followed by an `ADD_PARTITION` instead of leveraging the `REPLACE` option due to the efficient behavior of the `DROP_PARTITION` option.

#### 5.2.4.2.2 Add Partition

Oracle Data Mining supports adding a single partition or multiple partitions to an existing partitioned model.

The addition occurs based on the input data set and the name of the existing partitioned model. The operation takes the input data set and the existing partitioned model as parameters. The partition keys are extracted from the input data set and the model partitions are built against the input data set. These partitions are added to the partitioned model. In the case where partition keys for new partitions conflict with the existing partitions in the model, you can select from the following three approaches to resolve the conflicts:

- `ERROR`: Terminates the ADD operation without adding any partitions.
- `REPLACE`: Replaces the existing partition for which the conflicting keys are found.
- `IGNORE`: Eliminates the rows having the conflicting keys.

If the input data set contains multiple keys, then the operation creates multiple partitions. If the total number of partitions in the model increases to more than the user-defined maximum specified when the model was created, then you get an error. The default threshold value for the number of partitions is `1000`.

### 5.2.4.3 Partitioned Model scoring

Learn about scoring of a partitioned model.

The scoring of the partitioned model is the same as that of the non-partitioned model. The syntax of the data mining function remains the same but is extended to provide an optional hint to you. The optional hint can impact the performance of a query which involves scoring a partitioned model.

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

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

**Related Topics**

- *Oracle Database SQL Language Reference*

## 5.3 Specifying Model Settings

Understand how to configure data mining models at build time.

Numerous configuration settings are available for configuring data mining models at build time. To specify settings, create a settings table with the columns shown in the following table and pass the table to `CREATE_MODEL`.

**Table 5-4 Settings Table Required Columns**

| Column Name | Data Type |
|---|---|
| `setting_name` | `VARCHAR2(30)` |

**Table 5-4 (Cont.) Settings Table Required Columns**

| Column Name | Data Type |
|---|---|
| setting_value | VARCHAR2(4000) |

Example 5-1 creates a settings table for an 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.

Some settings apply generally to the model, others are specific to an algorithm. Model settings are referenced in Table 5-5 and Table 5-6.

**Table 5-5 General Model Settings**

| Settings | Description |
|---|---|
| Mining function settings | See "Mining Function Settings" in *Oracle Database PL/SQL Packages and Types Reference* |
| Algorithm names | See "Algorithm Names" in *Oracle Database PL/SQL Packages and Types Reference* |
| Global model characteristics | See "Global Settings" in *Oracle Database PL/SQL Packages and Types Reference* |
| Automatic Data Preparation | See "Automatic Data Preparation" in *Oracle Database PL/SQL Packages and Types Reference* |

**Table 5-6 Algorithm-Specific Model Settings**

| Algorithm | Description |
|---|---|
| CUR Matrix Decomposition | See "DBMS_DATA_MINING —Algorithm Settings: CUR Matrix Decomposition"in *Oracle Database PL/SQL Packages and Types Reference* |
| Decision Tree | See "DBMS_DATA_MINING —Algorithm Settings: Decision Tree" in *Oracle Database PL/SQL Packages and Types Reference* |
| Expectation Maximization | See "DBMS_DATA_MINING —Algorithm Settings: Expectation Maximization" in *Oracle Database PL/SQL Packages and Types Reference* |
| Explicit Semantic Analysis | See “DBMS_DATA_MINING —Algorithm Settings: Explicit Semantic Analysis” in *Oracle Database PL/SQL Packages and Types Reference* |
| Exponential Smoothing | See "DBMS_DATA_MINING —Algorithm Settings: Exponential Smoothing Models" in *Oracle Database PL/SQL Packages and Types Reference* |
| Generalized Linear Models | See "DBMS_DATA_MINING —Algorithm Settings: Generalized Linear Models" in *Oracle Database PL/SQL Packages and Types Reference* |
| *k*-Means | See "DBMS_DATA_MINING —Algorithm Settings: *k*-Means" in *Oracle Database PL/SQL Packages and Types Reference* |
| Naive Bayes | See "Algorithm Settings: Naive Bayes" in *Oracle Database PL/SQL Packages and Types Reference* |
| Neural Network | See "DBMS_DATA_MINING —Algorithm Settings: Neural Network" in *Oracle Database PL/SQL Packages and Types Reference* |

**Table 5-6 (Cont.) Algorithm-Specific Model Settings**

| Algorithm | Description |
|---|---|
| Non-Negative Matrix Factorization | See "DBMS_DATA_MINING —Algorithm Settings: Non-Negative Matrix Factorization" in *Oracle Database PL/SQL Packages and Types Reference* |
| O-Cluster | See "Algorithm Settings: O-Cluster" in *Oracle Database PL/SQL Packages and Types Reference* |
| Random Forest | See "DBMS_DATA_MINING — Algorithm Settings: Random Forest" in *Oracle Database PL/SQL Packages and Types Reference* |
| Singular Value Decomposition | See "DBMS_DATA_MINING —Algorithm Settings: Singular Value Decomposition" in *Oracle Database PL/SQL Packages and Types Reference* |
| Support Vector Machine | See "DBMS_DATA_MINING —Algorithm Settings: Support Vector Machine" in *Oracle Database PL/SQL Packages and Types Reference* |

**Example 5-1 Creating a Settings Table for an SVM Classification Model**

```
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;
/
```

**Related Topics**

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

## 5.3.1 Specifying 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 5-7 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 5-7, 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 Data Mining Concepts*

## 5.3.2 Specifying 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 5-8 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 data mining and understand the different target data types.
- *Oracle Data Mining Concepts*

## 5.3.3 Specifying Class Weights

Specify class weights table settings in Logistic Regression or Support Vector Machine (SVM) Classification to favour 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 5-9 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 data mining and understand the different target data types.

- *Oracle Data Mining Concepts*

## 5.3.4 Model Settings in the Data Dictionary

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

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

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

```
SQL> describe all_mining_model_settings
 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 5-10 ALL_MINING_MODEL_SETTINGS**

| Column | Description |
|---|---|
| owner | Owner of the mining model. |
| model_name | Name of the mining 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.

**Example 5-2 ALL_MINING_MODEL_SETTINGS**

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

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*

13

## 5.3.5 Specifying Mining Model Settings for R Model

The mining model settings for R model determine the characteristics of the model. You can specify the mining model settings in the `mining_model_table`.

You can build R models with the mining model settings by combining together generic settings that do not require an algorithm, such as `ODMS_PARTITION_COLUMNS` and `ODMS_SAMPLING`. The following settings are exclusive to R mining model, and they allow you to specify the R Mining model:

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

**Related Topics**

- Registered R Scripts
  The `RALG_*_FUNCTION` must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

### 5.3.5.1 ALGO_EXTENSIBLE_LANG

Use the `ALGO_EXTENSIBLE_LANG` setting to specify the Oracle Data Mining framework with extensible algorithms.

Currently, `R` is the only valid value for `ALGO_EXTENSIBLE_LANG`. When the value for `ALGO_EXTENSIBLE_LANG` is set to R, the mining models are built using the R language. You can use the following settings in the `model_setting_table` to specify the build, score, and view 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` must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

### 5.3.5.2 RALG_BUILD_FUNCTION

Use the `RALG_BUILD_FUNCTION` to specify the name of an existing registered R script for R algorithm mining model build.

You must specify both `RALG_BUILD_FUNCTION` and `ALGO_EXTENSIBLE_LANG` in the `model_setting_table`. The R script defines an R function that has the first input argument of `data.frame` for training data, and it returns an R 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 5-3 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 in the `model_setting_table`.

```
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 R script repository which uses the existing ORE security restrictions. You can use Oracle R Enterprise API `sys.rqScriptCreate` to register the script. Oracle R Enterprise 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 mining function model build, 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. R attribute `dm$nclus` is set with the number of clusters for the return 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. R attribute `dm$nfeat` is set with the number of features for the return feature extraction model.

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

**Related Topics**

- RALG_BUILD_PARAMETER
  The `RALG_BUILD_FUNCTION` input parameter specifies a list of numeric and string scalar values in SQL `SELECT` query statement format.
- Registered R Scripts
  The `RALG_*_FUNCTION` must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

## 5.3.5.2.1 RALG_BUILD_PARAMETER

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

**Example 5-4 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 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 `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 to specify the name of an existing registered R script for R algorithm mining model build.

## 5.3.5.3 RALG_DETAILS_FUNCTION

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

Use the RALG_DETAILS_FUNCTION to specify an existing registered R script that generates model information. The specified R 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 RALG_DETAILS_FORMAT, and can contain only numeric or string scalar types.

**Example 5-5 Example of RALG_DETAILS_FUNCTION**

This example shows how to specify the name of the R script MY_LM_DETAILS_SCRIPT in the model_setting_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 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 must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.
- RALG_DETAILS_FORMAT
  Use the RALG_DETAILS_FORMAT parameter to specify the names and column types in the model view. It is a string that contains a SELECT query to specify a list of numeric and string scalar data types for the name and type of the model view columns.

### 5.3.5.3.1 RALG_DETAILS_FORMAT

Use the RALG_DETAILS_FORMAT parameter to specify the names and column types in the model view. It is a string that contains a SELECT query to specify a list of numeric and string scalar data types for the name and type of the model view columns.

When RALG_DETAILS_FORMAT and RALG_DETAILS_FUNCTION 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 NULL value for non-partitioned models. The other columns of the model view are defined by RALG_DETATLS_FORMAT.

**Example 5-6 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 `varchar2` column `attr_name` and 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 `data.frame`.

## 5.3.5.4 RALG_SCORE_FUNCTION

Use the `RALG_SCORE_FUNCTION` to specify an existing registered R script for R algorithm mining model score in the `mining_model_table`.

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

**Example 5-7 Example of RALG_SCORE_FUNCTION**

This example shows how the function takes the R model and scores the data in the `data.frame`. The argument object is the R Linear Model. The argument `newdata` contains scoring data in the `data.frame`.

```
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])}
```

In this example,

- `object` indicates the LM model
- `newdata` indicates the scoring data.frame

The output of the specified 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 mining techniques, such as:

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

**Example 5-8 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_setting_table`.

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

In the 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 5-9 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_setting_table`.

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

In the R script repository, *MY_LOGITGLM_PREDICT_SCRIPT* is registered as follows. It is a logit Classification with two target class "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 extra column containing the value of the distance to each cluster in order of cluster ID after the columns for the predicted probability.

**Example 5-10 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_setting_table`.

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

```
End;
/
```

In the R 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, and they are consecutive values starting from 1.

**Example 5-11 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_setting_table`.

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

In the R 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` must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

## 5.3.5.5 RALG_WEIGHT_FUNCTION

Use the `RALG_WEIGHT_FUNCTION` to specify the name of an existing registered R script that computes weight or contribution for each attribute in scoring. The specified R

script is used in the query function PREDICTION_DETAILS to evaluate attribute contribution.

The specified R script defines an R function containing the first input argument for model object, and the second input argument of data.frame for scoring data. When the mining function is Classification, Clustering, or Feature Extraction, the target class name or cluster ID or feature ID is passed by the third input argument to compute the weight for that particular class or 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 5-12 Example of RALG_WEIGHT_FUNCTION**

This example shows how to specify the name of the R script *MY_PREDICT_WEIGHT_SCRIPT* that computes 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 R script repository, the script *MY_PREDICT_WEIGHT_SCRIPT* for Regression is registered as:

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

In the R 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 must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

## 5.3.5.6 Registered R Scripts

The RALG_*_FUNCTION must specify R scripts that exist in the R script repository. You can register the R scripts using Oracle R Enterprise.

The RALG_*_FUNCTION includes the following functions:

- RALG_BUILD_FUNCTION
- RALG_DETAILS_FUNCTION
- RALG_SCORE_FUNCTION
- RALG_WEIGHT_FUNCTION

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

You can register the R scripts through Oracle Enterprise R (ORE). To register R scripts, you must have the `RQADMIN` role. After an R model is built, the names of these specified R scripts become model settings. These R scripts must exist in the R 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 Oracle Enterprise R as well.

### 5.3.5.7 R Model Demonstration Scripts

You can access R model demonstration scripts under `rdbms/demo`

```
dmraidemo.sql  dmrglmdemo.sql dmrpcademo.sql
dmrardemo.sql  dmrkmdemo.sql  dmrrfdemo.sql
dmrdtdemo.sql  dmrnndemo.sql
```

### 5.3.5.8 Algorithm Meta Data Registration

Algorithm Meta Data Registration allows for a uniform and consistent approach of registering new techniques and their settings.

User have the ability to add new algorithms through the registration process. The new algorithms can appear as available within Oracle Data Mining R within their appropriate mining techniques. Based on the registration meta data, the settings page is dynamically rendered. Algorithm meta data registration extends the mining model capability of Oracle Data Mining.

**Related Topics**

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

## 5.4 Model Detail Views

The `GET_*` interfaces are replaced by model views, and Oracle recommends that users leverage the views instead.

The following are the new model views:

**Association**:

- Model Detail Views for Association Rules
- Model Detail View for Frequent Itemsets
- Model Detail View for Transactional Itemsets
- Model Detail View for Transactional Rule

**Classification, Regression, and Anomaly Detection**:

- Model Detail Views for Classification Algorithms
- Model Detail Views for CUR Matrix Decomposition
- Model Detail Views for Decision Tree
- Model Detail Views for Generalized Linear Model
- Model Detail Views for Naive Bayes
- Model Detail Views for Neural Network
- Model Detail Views for Random Forest
- Model Detail View for Support Vector Machine

**Clustering**:

- Model Detail Views for Clustering Algorithms
- Model Detail Views for Expectation Maximization
- Model Detail Views for *k*-Means
- Model Detail Views for O-Cluster

**Feature Extraction**:

- Model Detail Views for Explicit Semantic Analysis
- Model Detail Views for Non-Negative Matrix Factorization
- Model Detail Views for Singular Value Decomposition

**Feature Selection**:

- Model Detail View for Minimum Description Length

**Data Preparation and Other**:

- Model Detail View for Binning
- Model Detail Views for Global Information
- Model Detail View for Normalization and Missing Value Handling

**Time Series**:

Model Detail Views for Exponential Smoothing Models

## 5.4.1 Model Detail Views for Association Rules

Model detail views for Association Rules describe the rule view for Association Rules. Oracle recommends that users leverage the model details views instead of the `GET_ASSOCIATION_RULES` function.

The rule view `DM$VR`*model_name* describes the generated rules for Association Rules. Depending on the settings of the model, the rule view has different set 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 setting `ITEM_NAME` (`ODMS_ITEM_ID_COLUMN_NAME`) is set and `ITEM_VALUE` (`ODMS_ITEM_VALUE_COLUMN_NAME`) is not set, the following is the view. Here the consequent item is defined with only name field. If `ITEM_VALUE` setting is also set, the view will have one extra column `CONSEQUENT_VALUE` to specify 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 5-11 Rule View Columns for Transactional Inputs**

| Column Name | Description |
|---|---|
| `PARTITION_NAME` | A partition in a partitioned model to retrieve details |
| `RULE_ID` | Identifier of the rule |

**Table 5-11 (Cont.) Rule View Columns for Transactional Inputs**

| Column Name | Description |
|---|---|
| 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 | Name of the consequent |
| CONSEQUENT_VALUE | Value of the consequent when setting Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as numerical, the view has a CONSEQUENT_VALUE column.<br>When setting Item_value (ODMS_ITEM_VALUE_COLUMN_NAME) is set with TYPE as categorical, the view has a CONSEQUENT_VALUE column. |
| 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.<br>• When setting ODMS_ITEM_VALUE_COLUMN_NAME is not set, each item is defined by item_name. As an example, assume the antecedent contains one item B, it is represented as follows:<br><itemset NUMAGGR="0"><item><item_name>B</item_name></item></itemset><br>As another example, assume the antecedent contains two items, A and C, it is represented as follows:<br><itemset NUMAGGR="0"><item><item_name>A</item_name></item><item><item_name>C</item_name></item></itemset><br>• When setting ODMS_ITEM_VALUE_COLUMN_NAME is set, each item is defined by item_name and item_value. As an example, assume the antecedent contains two items, (name A, value 1) and (name C, value 1), then it is represented as follows:<br><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 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 example, refer “Example: Calculating Aggregates” in *Oracle Data Mining 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 5-13 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
```

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.

#### 2-Dimensional Inputs

In Oracle Data Mining, 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 input data table contains nested table. In such case, the `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 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 the types for three parts 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 which are specific to 2-D inputs.

**Table 5-12 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` | Value of the consequent when setting `Item_value` is set with `TYPE` as numerical, the view has a `CONSEQUENT_VALUE` column.<br>When setting `Item_value` is set with `TYPE` as categorical, the view has a `CONSEQUENT_VALUE` column. |
| `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`:<br>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:<br>`<itemset NUMAGGR="0"><item><item_name>ADDR</item_name><item_subname></item_subna me><item_value>MA</item_value></item></itemset>`<br>For 2-Dimensional input with nested table, the subname field is filled. |

### Global Detail for Association Rules

A single global detail is produced by an Association model. The following table describes a global detail returned for Association Rules model.

**Table 5-13 Global Detail for Association Rules**

| 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 input data |

## 5.4.2 Model Detail View for Frequent Itemsets

Model detail view for Frequent Itemsets describes the frequent itemsets view. Oracle recommends that you leverage model details view instead of the `GET_FREQUENT_ITEMSETS` function.

The frequent itemsets view `DM$VI`*model_name* has the following schema:

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

**Table 5-14 Frequent Itemsets View**

| Column Name | Description |
|---|---|
| PARTITION_NAME | A partition in a partitioned model |
| ITEMSET_ID | Itemset identifier |
| SUPPORT | Support of the itemset |
| NUMBER_OF_ITEMS | Number of items in the itemset |
| ITEMSET | Frequent itemset<br>The structure of the `SYS.XMLTYPE` column itemset is the same as the corresponding Antecedent column of the rule view. |

## 5.4.3 Model Detail View for Transactional Itemsets

Model detail view for Transactional Itemsets describes the transactional itemsets view. Oracle recommends that users leverage the model details views.

For the very common case of transactional data without aggregates, `DM$VT`*model_name* view 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 schema:

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

**Table 5-15 Transactional Itemsets 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 |

## 5.4.4 Model Detail View for Transactional Rule

Model detail view for Transactional Rule describes the transactional rule view and transactional itemsets view. Oracle recommends that you leverage model details views.

Transactional data without aggregates also has a transactional rule 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 schema:

```
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 5-16 Transactional Rule 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 |

## 5.4.5 Model Detail Views for Classification Algorithms

Model detail view for Classification algorithms describe target map view and scoring cost view which are applicable to all Classification algorithms. Oracle recommends that users leverage the model details views instead of the `GET_*` function.

The target map view `DM$VT`*model_name* describes the target distribution for Classification models. The view has the following schema:

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

**Table 5-17 Target Map 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 view DM$VC*model_name* describes the scoring cost matrix for Classification models. The view has the following schema:

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

**Table 5-18 Scoring Cost View**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| ACTUAL_TARGET_VALUE | A valid target value |
| PREDICTED_TARGET_VALUE | Predicted target value |
| COST | Associated cost for the actual and predicted target value pair |

## 5.4.6 Model Detail Views for CUR Matrix Decomposition

Model Detail Views for CUR Matrix Decomposition describe scores and ranks of attributes and rows.

CUR Matrix Decomposition algorithm has 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 schema:

```
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 5-19 Attribute Importance and Rank View**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| ATTRIBUTE_NAME | Attribute name |

**Table 5-19 (Cont.) Attribute Importance and Rank View**

| Column Name | Description |
|---|---|
| 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 schema:

```
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 5-20 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 5-21 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 |

## 5.4.7 Model Detail Views for Decision Tree

Model detail view for Decision Tree describes the split information view, node statistics view, node description view, and the cost matrix view. Oracle recommends that users leverage the model details views instead of `GET_MODEL_DETAILS_XML` function.

The split information 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 schema:

```
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 5-22 Split Information 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.<br>For example, `<Element>Windy</Element><Element>Hot</Element>`. |

The node 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 schema:

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

```
TARGET_VALUE                   NUMBER/VARCHAR2
TARGET_SUPPORT                 NUMBER
```

**Table 5-23 Node 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 |

Higher level node description can be found in DM$VO*model_name* view. The DM$VO*model_name* has the following schema:

```
ame                                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 5-24 Node Description 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:<br>*IN*, = , <>, < , >, <=, and >= |
| VALUE | Value used as the description criterion. This is an XML element described using the <Element> tag.<br>For example, <Element>Windy</Element><Element>Hot</Element>. |

The DM$VM*model_name* view describes the cost matrix used by the Decision Tree build. The DM$VM*model_name* view has the following schema:

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

**Table 5-25 Cost Matrix View**

| Parameter | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| ACTUAL_TARGET_VALUE | Valid target value |
| PREDICTED_TARGET_VALUE | Predicted Target value |
| COST | Associated cost for the actual and predicted target value pair |

The following table describes the global view for Decision Tree.

**Table 5-26 Decision Tree Statistics Information In Model Global View**

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

## 5.4.8 Model Detail Views for Generalized Linear Model

Model details views for Generalized Linear Model (GLM) describes the model details view and row diagnostic view for Linear and Logistic Regression. Oracle recommends that users leverage model details views than the GET_MODEL_DETAILS_GLM function.

The model details 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 schema:

```
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 schema:

```
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 5-27 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 mining 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.<br>When the nested column is numeric, the mining 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 mining attribute is identified by a unique ATTRIBUTE_NAME.ATTRIBUTE_VALUE pair. For nested categorical columns, a mining attribute is identified by the combination: ATTRIBUTE_NAME.ATTRIBUTE_SUBNAME.ATTRIBUTE_VALUE. For numerical attributes, ATTRIBUTE_VALUE is null. |

**Table 5-27 (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 simply the fully-qualified attribute name (*attribute_name.attribute_subname* if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form:<br>*fully-qualified_attribute_name.attribute_value*<br>When feature generation is enabled, a term in the model can be a single mining attribute or the product of up to 3 mining attributes. Component mining 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.<br>**Note:**<br>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.<br>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 row diagnostic 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 schema:

```
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 5-28 Row Diagnostic 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 schema:

```
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 5-29 Row Diagnostic 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.<br>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.<br>RESIDUAL is 1 minus the predicted probability of the actual target value for the row. |
| DEVIANCE_RESIDUAL | The DEVIANCE_RESIDUAL is the contribution to the model deviance of the $i^{th}$ observation. For Logistic Regression it has the form the square root of 2 times the log(1 + e^eta) - eta for the non-reference class and -square root of 2 time the log (1 + eta) for the reference class, where eta is the linear prediction (the prediction as if the model were a Linear Regression). |

**Table 5-29 (Cont.) Row Diagnostic View for Logistic Regression**

| Column Name | Description |
|---|---|
| C | Measures the overall change in the fitted logits due to the deletion of the $i^{th}$ observation for all points including the one deleted (the $i^{th}$ point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by the square of 1 minus the Hat value.<br>Confidence interval displacement diagnostics that provides scalar measure of the influence of individual observations. |
| CBAR | C and CBAR are extensions of Cooks' distance for Logistic Regression. CBAR measures the overall change in the fitted logits due to the deletion of the $i^{th}$ observation for all points excluding the one deleted (the $i^{th}$ point). It is computed as the square of the Pearson residual multiplied by the Hat value divided by (1 minus the Hat value)<br>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 details returned by a Linear Regression model.

**Table 5-30 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:<br>• YES<br>• NO |
| CORRECTED_TOTAL_DF | Corrected total degrees of freedom |
| CORRECTED_TOT_SS | Corrected total sum of squares |
| DEPENDENT_MEAN | Dependent mean |
| ERROR_DF | Error degrees of freedom |
| ERROR_MEAN_SQUARE | Error mean square |
| ERROR_SUM_SQUARES | Error sum of squares |
| F_VALUE | Model *F* value statistic |
| GMSEP | Estimated mean square error of the prediction, assuming multivariate normality |

**Table 5-30 (Cont.) Global Details for Linear Regression**

| Name | Description |
|---|---|
| HOCKING_SP | Hocking Sp statistic |
| ITERATIONS | Tracks the number of SGD iterations. Applicable only when the solver is SGD. |
| J_P | JP statistic (the final prediction error) |
| MODEL_DF | Model degrees of freedom |
| MODEL_F_P_VALUE | Model *F* value probability |
| MODEL_MEAN_SQUARE | Model mean square error |
| MODEL_SUM_SQUARES | Model sum of square errors |
| NUM_PARAMS | Number of parameters (the number of coefficients, including the intercept) |
| NUM_ROWS | Number of rows |
| R_SQ | R-Square |
| RANK_DEFICIENCY | The number of predictors excluded from the model due to multi-collinearity |
| ROOT_MEAN_SQ | Root mean square error |
| SBIC | Schwarz's Bayesian information criterion |

### Global Details for GLM: Logistic Regression

The following table returns global details returned by a Logistic Regression model.

**Table 5-31 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:<br>• YES<br>• NO |
| DEPENDENT_MEAN | Dependent mean |
| ITERATIONS | Tracks the number of SGD iterations (number of IRLS iterations). Applicable only when the solver is SGD. |
| LR_DF | Likelihood ratio degrees of freedom |
| LR_CHI_SQ | Likelihood ratio chi-square value |
| LR_CHI_SQ_P_VALUE | Likelihood ratio chi-square probability value |
| NEG2_LL_INTERCEPT | -2 log likelihood of the baseline, intercept-only, model |
| NEG2_LL_MODEL | -2 log likelihood of the model |

**Table 5-31 (Cont.) Global Details for Logistic Regression**

| Name | Description |
|---|---|
| NUM_PARAMS | Number of parameters (the number of coefficients, including the intercept) |
| NUM_ROWS | Number of rows |
| PCT_CORRECT | Percent of correct predictions |
| PCT_INCORRECT | Percent of incorrectly predicted rows |
| PCT_TIED | Percent of cases where the estimated probabilities are equal for both target classes |
| PSEUDO_R_SQ_CS | Pseudo R-square Cox and Snell |
| PSEUDO_R_SQ_N | Pseudo R-square Nagelkerke |
| RANK_DEFICIENCY | The number of predictors excluded from the model due to multi-collinearity |
| SC_INTERCEPT | Schwarz's Criterion for the fit of the baseline, intercept-only, model |
| SC_MODEL | Schwarz's Criterion for the fit of the intercept and the covariates (predictors) model |

**Note:**

- When Ridge Regression is enabled, fewer global details are returned. For information about ridge, see *Oracle Data Mining 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 describes global statistics view, alert view, and computed settings view. Oracle recommends that users leverage the model details views instead of GET_MODEL_DETAILS_GLOBAL function.

## 5.4.9 Model Detail Views for Naive Bayes

Model Detail Views for Naive Bayes describes prior view and result view. Oracle recommends that users leverage the model details views instead of the GET_MODEL_DETAILS_NB function.

The prior view DM$VP*model_name* describes the priors of the targets for Naïve Bayes. The view has the following schema:

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

```
TARGET_VALUE                    NUMBER/VARCHAR2
PRIOR_PROBABILITY               BINARY_DOUBLE
COUNT                           NUMBER
```

**Table 5-32 Prior 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 Naïve Bayes result view DM$VV*model_view* describes the conditional probabilities of the Naïve Bayes model. The view has the following schema:

```
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 5-33 Result 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 | Mining attribute value for the column ATTRIBUTE_NAME or the nested column ATTRIBUTE_SUBNAME (if any). |
| CONDITIONAL_PROBABILITY | Conditional probability of a mining attribute for a given target |
| COUNT | Number of rows for a given mining attribute and a given target |

The following table describes the global view for Naive Bayes.

**Table 5-34 Naive Bayes Statistics Information In Model Global View**

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

## 5.4.10 Model Detail Views for Neural Network

Model Detail Views for Neural Network describes the weights of the neurons: input layer and hidden layers. Oracle recommends that users leverage the model details views.

Neural Network algorithm has the following views:

Weights: DM$VA*model_name*

The view DM$VA*model_name* has the following schema:

```
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 5-35 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 DM$VG*model_name* is a pre-existing view. The following name-value pairs are added to the view.

**Table 5-36 Neural Networks Statistics Information In Model Global View**

| Name | Description |
|---|---|
| CONVERGED | Indicates whether the model build process has converged to specified tolerance. The following are the possible values:<br>• YES<br>• 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) |

## 5.4.11 Model Detail Views for Random Forest

Model Detail Views for Random Forest describes variable importance measures and statistics in global view. Oracle recommends that users leverage the model details views.

Random Forest algorithm has the following statistics views:

- Variable importance statistics DM$VA*model_name*
- Random Forest statistics in model global view DM$VG*model_name*

One of the important outputs from the Random Forest model build is a ranking of attributes based on their relative importance. This is measured using Mean Decrease Gini. The view DM$VA*model_name* has the following schema:

```
Name                           Type
-----------------------        ---------------
PARTITION_NAME                 VARCHAR2(128)
ATTRIBUTE_NAME                 VARCHAR2(128)
ATTRIBUTE_SUBNAME              VARCHAR2(128)
ATTRIBUTE_IMPORTANCE           BINARY_DOUBLE
```

**Table 5-37 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 view DM$VG*model_name* is a pre-existing view. The following name-value pairs are added to the view.

**Table 5-38 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 |

## 5.4.12 Model Detail View for Support Vector Machine

Model Detail View for Support Vector Machine describes linear coefficient view. Oracle recommends that users leverage the model details views instead of the `GET_MODEL_DETAILS_SVM` function.

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 schema:

```
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 5-39 Linear Coefficient View for Support Vector Machine**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| TARGET_VALUE | Target value, numerical or categorical |
| ATTRIBUTE_NAME | Column name |
| ATTRIBUTE_SUBNAME | Nested column subname. The value is null for non-nested columns. |
| ATTRIBUTE_VALUE | Value of a categorical attribute |
| COEFFICIENT | Projection coefficient value |
| REVERSED_COEFFICIENT | Coefficient transformed on the original scale |

The following table describes the Support Vector statistics global view.

**Table 5-40 Support Vector Statistics Information In Model Global View**

| Name | Description |
|---|---|
| CONVERGED | Indicates whether the model build process has converged to specified tolerance:<br>• YES<br>• 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. |

## 5.4.13 Model Detail Views for Clustering Algorithms

Oracle Data Mining supports these clustering algorithms: Expectation Maximization, *k*-Means, and Orthogonal Partitioning Clustering (O-Cluster).

All clustering algorithms share the following views:

- Cluster description DM$VD*model_name*
- Attribute statistics DM$VA*model_name*
- Histogram statistics DM$VH*model_name*
- Rule statistics DM$VR*model_name*

The cluster description view DM$VD*model_name* describes cluster level information about a clustering model. The view has the following schema:

```
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 5-41 Cluster Description View for Clustering Algorithm**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| CLUSTER_ID | The ID of a cluster in the model |
| CLUSTER_NAME | Specifies the label of the cluster |
| RECORD_COUNT | Specifies the number of records |
| PARENT | The ID of the parent |

**Table 5-41 (Cont.) Cluster Description View for Clustering Algorithm**

| Column Name | Description |
|---|---|
| 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 schema:

```
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 5-42 Attribute View for Clustering Algorithm**

| Column Name | Description |
|---|---|
| PARTITION_NAME | A partition in a partitioned model |
| CLUSTER_ID | The ID of a cluster in the model |
| CLUSTER_NAME | Specifies the label of the cluster |
| ATTRIBUTE_NAME | Specifies the attribute name |
| ATTRIBUTE_SUBNAME | Specifies the attribute subname |
| MEAN | The field returns the average value of a numeric attribute |
| VARIANCE | The variance of a numeric attribute |
| 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 schema:

```
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 5-43 Histogram View for Clustering Algorithm**

| 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 schema:

```
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 5-44 Rule View for Clustering Algorithm**

| 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 |

**Table 5-44 (Cont.) Rule View for Clustering Algorithm**

| Column Name | Description |
|---|---|
| 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 |

## 5.4.14 Model Detail Views for Expectation Maximization

Model detail views for Expectation Maximization (EM) describes the differences in the views for EM against those of Clustering views. Oracle recommends that user leverage the model details views instead of the `GET_MODEL_DETAILS_EM` function.

The following views are the differences in the views for Expectation Maximization against Clustering views. For an overview of the different Clustering views, refer to "Model Detail Views for Clustering Algorithms".

The component view `DM$VO`*model_name* describes the EM components. The component view contains information about their prior probabilities and what cluster they map to. The view has the following schema:

```
Name                                      Type
 ----------------------------------- ----------------------------
 PARTITION_NAME                           VARCHAR2(128)
 COMPONENT_ID                             NUMBER
 CLUSTER_ID                               NUMBER
 PRIOR_PROBABILITY                        BINARY_DOUBLE
```

**Table 5-45 Component 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 mean and variance component 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 schema:

```
Name                                      Type
 ----------------------------------- ----------------------------
```

```
 PARTITION_NAME                 VARCHAR2(128)
 COMPONENT_ID                   NUMBER
 ATTRIBUTE_NAME                 VARCHAR2(4000)
 MEAN                           BINARY_DOUBLE
 VARIANCE                       BINARY_DOUBLE
```

The frequency component 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 schema:

```
Name                           Type
 ------------------------------ --------------------------
 PARTITION_NAME                 VARCHAR2(128)
 COMPONENT_ID                   NUMBER
 ATTRIBUTE_NAME                 VARCHAR2(4000)
 ATTRIBUTE_VALUE                VARCHAR2(4000)
 FREQUENCY                      BINARY_DOUBLE
```

**Table 5-46 Frequency Component 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 DM$VI*model_name* view and has the following schema:

```
Name                                    Type
 ---------------------------------------- --------------------------
 PARTITION_NAME                          VARCHAR2(128)
 ATTRIBUTE_NAME                          VARCHAR2(128)
 ATTRIBUTE_IMPORTANCE_VALUE              BINARY_DOUBLE
 ATTRIBUTE_RANK                          NUMBER
```

**Table 5-47 2–Dimensional Attribute Ranking for Expectation Maximization**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| ATTRIBUTE_NAME | Column name |
| ATTRIBUTE_IMPORTANCE_VALUE | Importance value |

**Table 5-47 (Cont.) 2–Dimensional Attribute Ranking for Expectation Maximization**

| Column Name | Description |
|---|---|
| ATTRIBUTE_RANK | An attribute rank based on the importance value |

The `pairwise` Kullback–Leibler divergence is reported in the `DM$VB`*model_name* view. 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 schema:

```
Name                                      Type
 ----------------------------------------- ---------------------------
 PARTITION_NAME                            VARCHAR2(128)
 ATTRIBUTE_NAME_1                          VARCHAR2(128)
 ATTRIBUTE_NAME_2                          VARCHAR2(128)
 DEPENDENCY                                BINARY_DOUBLE
```

**Table 5-48 Kullback-Leibler Divergence for Expectation Maximization**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| ATTRIBUTE_NAME_1 | Name of an attribute 1 |
| ATTRIBUTE_NAME_2 | Name of an attribute 2 |
| 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 schema:

```
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 5-49 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 |

**Table 5-49 (Cont.) Projection table for Expectation Maximization**

| Column Name | Description |
|---|---|
| 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. |

### Global Details for Expectation Maximization

The following table describes global details for Expectation Maximization.

**Table 5-50 Global Details for Expectation Maximization**

| Name | Description |
|---|---|
| CONVERGED | Indicates whether the model build process has converged to specified tolerance. The possible values are:<br>• YES<br>• 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 |
| 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 Data Mining supports these clustering algorithms: Expectation Maximization, *k*-Means, and Orthogonal Partitioning Clustering (O-Cluster).

## 5.4.15 Model Detail Views for *k*-Means

Model detail views for *k*-Means (KM) describes cluster description view and scoring view. Oracle recommends that you leverage model details view instead of GET_MODEL_DETAILS_KM function.

This section describes the differences in the views for *k*-Means against the Clustering views. For an overview of the different views, refer to "Model Detail Views for Clustering Algorithms". For *k*-Means, the cluster description view DM$VD*model_name* has an additional column:

```
Name                              Type
 --------------------------------- ----------------------------
 DISPERSION                        BINARY_DOUBLE
```

**Table 5-51 Cluster 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 scoring 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 5-52 Scoring View for k-Means**

| 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 view for *k*–Means.

**Table 5-53 *k*–Means Statistics Information In Model Global View**

| Name | Description |
|---|---|
| CONVERGED | Indicates whether the model build process has converged to specified tolerance. The following are the possible values:<br>• YES<br>• 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 Data Mining supports these clustering algorithms: Expectation Maximization, *k*-Means, and Orthogonal Partitioning Clustering (O-Cluster).

## 5.4.16 Model Detail Views for O-Cluster

Model Detail Views for O-Cluster describes the statistics views. Oracle recommends that user leverage the model details views instead of the `GET_MODEL_DETAILS_OC` function.

The following are the differences in the views for O-Cluster against Clustering views. For an overview of the different 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:

- Cluster description `DM$VD`*model_name*
- Attribute statistics `DM$VA`*model_name*
- Rule statistics `DM$VR`*model_name*
- Histogram statistics `DM$VH`*model_name*

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 schema:

```
Name                                      Type
 ----------------------------------- ----------------------------
 ATTRIBUTE_NAME                           VARCHAR2(128)
 ATTRIBUTE_SUBNAME                        VARCHAR2(4000)
 OPERATOR                                 VARCHAR2(2)
 VALUE                                    SYS.XMLTYPE
```

**Table 5-54 Description View**

| 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 histogram view DM$VH*model_name* with a different schema than EM and k-Means (KM). The view has the following schema:

```
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 5-55 Histogram Component View**

| 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 view for O-Cluster.

**Table 5-56 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 Data Mining supports these clustering algorithms: Expectation Maximization, *k*-Means, and Orthogonal Partitioning Clustering (O-Cluster).

## 5.4.17 Model Detail Views for Explicit Semantic Analysis

Model Detail Views for Explicit Semantic Analysis (ESA) describes attribute statistics view and feature view. Oracle recommends that users leverage the model details view.

ESA algorithm has the following views:

- Explicit Semantic Analysis Matrix DM$VA*model_name*: This view has different schemas 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 for only Feature Extraction.

The view DM$VA*model_name* has the following schema for Feature Extraction:

```
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 5-57 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 attribute coefficients for all target classes.

The view DM$VA*model_name* has the following schema 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 5-58 Explicit Semantic Analysis Matrix for Classification**

| Column Name | Description |
|---|---|
| PARTITION_NAME | Partition name in a partitioned model |
| TARGET_VALUE | Value of the target |
| ATTRIBUTE_NAME | Column name |
| ATTRIBUTE_SUBNAME | Nested column subname. The value is null for non-nested columns. |
| ATTRIBUTE_VALUE | Categorical attribute value |

**Table 5-58 (Cont.) Explicit Semantic Analysis Matrix for Classification**

| Column Name | Description |
|---|---|
| COEFFICIENT | A measure of the weight of the attribute with respect to the feature |

The 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 schema:

```
Name                              Type
--------------------------------- ---------------------------
PARTITION_NAME                    VARCHAR2(128)
FEATURE_ID                        NUMBER/VARHCAR2, DATE, TIMESTAMP,
                                  TIMESTAMP WITH TIME ZONE,
                                  TIMESTAMP WITH LOCAL TIME ZONE
```

**Table 5-59 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 view for Explicit Semantic Analysis.

**Table 5-60 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 |

## 5.4.18 Model Detail Views for Non-Negative Matrix Factorization

Model detail views for Non-Negative Matrix Factorization (NMF) describes encoding H matrix view and H inverse matrix view. Oracle recommends that users leverage the model details views instead of the GET_MODEL_DETAILS_NMF function.

The NMF algorithm has two matrix content views:

- Encoding (H) matrix DM$VE*model_name*
- H inverse matrix DM$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 schema definition.

```
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 5-61 Encoding H Matrix View for Non-Negative Matrix Factorization**

| 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 5-62 Inverse H Matrix View for Non-Negative Matrix Factorization**

| 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 statistics for Non-Negative Matrix Factorization.

**Table 5-63 Non-Negative Matrix Factorization Statistics Information In Model Global View**

| Name | Description |
|---|---|
| CONV_ERROR | Convergence error |
| CONVERGED | Indicates whether the model build process has converged to specified tolerance. The following are the possible values:<br>• YES<br>• NO |
| ITERATIONS | Number of iterations performed during build |
| NUM_ROWS | Number of rows used in the build input dataset |
| SAMPLE_SIZE | Number of rows used by the build |

## 5.4.19 Model Detail Views for Singular Value Decomposition

Model detail views for Singular Value Decomposition (SVD) describes S Matrix view, right-singular vectors view, and left-singular vector view. Oracle recommends that users leverage the model details views instead of the GET_MODEL_DETAILS_SVD function.

The DM$VE*model_name* view 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 a 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 schema:

```
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 5-64 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 |

**Table 5-64 (Cont.) S Matrix View**

| Column Name | Description |
|---|---|
| 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<br>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.<br>This column is only present for SVD models with setting dbms_data_mining.svds_scoring_mode set to dbms_data_mining.svds_scoring_pca<br>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 SVD DM$VV*model_view* describes the right-singular vectors of SVD model. For a PCA model it describes the principal components (eigenvectors). 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)
 VALUE                               BINARY_DOUBLE
```

**Table 5-65 Right-singular Vectors of Singular Value Decomposition**

| 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 view DM$VU*model_name* describes the left-singular vectors of a 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 schema:

```
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 5-66 Left-singular Vectors of Singular Value Decomposition 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 a global detail for Singular Value Decomposition.

**Table 5-67 Global Details 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*

## 5.4.20 Model Detail View for Minimum Description Length

Model detail view for Minimum Description Length (for calculating Attribute Importance) describes Attribute Importance view. Oracle recommends that users leverage the model details views instead of the `GET_MODEL_DETAILS_AI` function.

The Attribute Importance view `DM$VA`*model_name* describes the Attribute Importance as well as the Attribute Importance rank. The view has the following schema:

```
Name                                      Type
 ----------------------------------------- ----------------------------
 PARTITION_NAME                            VARCHAR2(128)
 ATTRIBUTE_NAME                            VARCHAR2(128)
 ATTRIBUTE_SUBNAME                         VARCHAR2(4000)
 ATTRIBUTE_IMPORTANCE_VALUE                BINARY_DOUBLE
 ATTRIBUTE_RANK                            NUMBER
```

**Table 5-68 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 view for Minimum Description Length.

**Table 5-69 Minimum Description Length Statistics Information In Model Global View**

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

## 5.4.21 Model Detail View for Binning

The binning view `DM$VB` describes the bin boundaries used in the automatic data preparation.

The view has the following schema:

```
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 5-70 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 |

## 5.4.22 Model Detail Views for Global Information

Model detail views for Global Information describes global statistics view, alert view, and computed settings view. Oracle recommends that users leverage the model details views instead of `GET_MODEL_DETAILS_GLOBAL` function.

The global statistics 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 schema:

```
Name                              Type
 ------------------          --------------------
 PARTITION_NAME                   VARCHAR2(128)
 NAME                             VARCHAR2(30)
 NUMERIC_VALUE                    NUMBER
 STRING_VALUE                     VARCHAR2(4000)
```

**Table 5-71 Global Statistics 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 alert view `DM$VW`*model_name* lists alerts issued during the model build. The view has the following schema:

```
Name                              Type
 ------------------          ----------------------
 PARTITION_NAME                   VARCHAR2(128)
 ERROR_NUMBER                     BINARY_DOUBLE
 ERROR_TEXT                       VARCHAR2(4000)
```

**Table 5-72 Alert 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 schema:

```
Name                                      Type
 -----------------                 --------------------
 PARTITION_NAME                            VARCHAR2(128)
 SETTING_NAME                              VARCHAR2(30)
 SETTING_VALUE                             VARCHAR2(4000)
```

**Table 5-73 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 |

## 5.4.23 Model Detail View for Normalization and Missing Value Handling

The Normalization and Missing Value Handling View DM$VN describes the normalization parameters used in Automatic Data Preparation (ADP) and the missing value replacement when a NULL value is encountered. Missing value replacement applies only to the twodimensional columns and does not apply to the nested columns.

The view has the following schema:

```
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 5-74 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 |

## 5.4.24 Model Detail Views for Exponential Smoothing Models

Model Detail Views for Exponential Smoothing Model (ESM) describes the views for model output and global information. Oracle recommends that users leverage the model details views.

Exponential Smoothing Model algorithm has the following views:

Model output: DM$VP*model_name*

Model global information: DM$VG*model_name*

Model output: This view gives the result of ESM model. The output has a set of records such as partition, CASE_ID, value, prediction, lower, upper, and so on and ordered by partition and CASE_ID (time). Each partition has a separate smoothing model. For a given partition, for each time (CASE_ID) point that the input time series covers, the value is the observed or accumulated value at the time point, and the prediction is the one-step-ahead forecast at that time point. For each time point (future prediction) beyond the range of input time series, the value is NULL, and the prediction is the model forecast for that time point. Lower and upper are the lower bound and upper bound of the user specified confidence interval for the prediction.

Model global Information: This view gives the global information of the model along with the estimated smoothing constants, the 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 5-75 Exponential Smoothing Model Statistics Information In Model Global View**

| 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 |

**Table 5-75 (Cont.) Exponential Smoothing Model Statistics Information In Model Global View**

| Name | Description |
|---|---|
| BETA | Trend smoothing constant |
| BIC | Bayesian information criterion |
| GAMMA | Seasonal smoothing constant |
| INITIAL LEVEL | Model estimate of value one time interval prior to start of observed series |
| INITIAL SEASON i | Model estimate of seasonal effect for season *i* one time interval prior to start of observed series |
| INITIAL TREND | Model estimate of trend one time interval prior to start of observed series |
| MAE | Model mean absolute error |
| MSE | Model mean square error |
| PHI | Damping parameter |
| STD | Model standard error |
| SIGMA | Model standard deviation of residuals |

# 6
# Scoring and Deployment

Explains the scoring and deployment features of Oracle Data Mining.



## 6.1 About Scoring and Deployment

**Scoring** is the application of models to new data. In Oracle Data Mining, 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 mining 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)

Oracle Data Mining supports all of these deployment scenarios.

> **Note:**
>
> Oracle Data Mining 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 Data Mining Concepts*
- Exporting and Importing Mining Models
  You can export machine learning models to move models to a different Oracle Database instance, such as from a development database to a production database.

# 6.2 Using the Data Mining SQL Functions

Learn about the benefits of SQL functions in data mining.

The data mining SQL 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 data mining techniques produce a score for each row in the selection. The functions can apply a mining model schema object to compute the score, or they can score dynamically without a pre-defined model, as described in "Dynamic Scoring".

**Related Topics**

- Dynamic Scoring
- Scoring Requirements
- Table 2-4
- *Oracle Database SQL Language Reference*

## 6.2.1 Choosing the Predictors

The SQL functions used for scoring 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.

The query in Example 6-1 uses all the predictors.

The query in Example 6-2 uses only gender, marital status, occupation, and income as predictors.

The query in Example 6-3 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 6-1 Using All 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 *) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;

C        CNT    AVG_AGE
- ---------- ----------
F         25         38
M        213         43
```

**Example 6-2 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;

C        CNT    AVG_AGE
- ---------- ----------
F         30         38
M        186         43
```

**Example 6-3 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;

C        CNT    AVG_AGE
- ---------- ----------
F         30         38
M        186         43
```

## 6.2.2 Single-Record Scoring

The functions used for data mining can produce a score for a single record, as shown in Example 6-4 and Example 6-5.

Example 6-4 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.

Example 6-5 returns a prediction for 'Affinity card is great' as the comments attribute by applying the text mining model `T_SVM_Clas_sample`. The resulting score is 1, meaning that this customer is likely to use an affinity card.

**Example 6-4 Scoring a Single Customer or a Single Text Expression**

```
SELECT PREDICTION (NB_SH_Clas_Sample USING *)
    FROM sh.customers where cust_id = 102001;

PREDICTION(NB_SH_CLAS_SAMPLEUSING*)
-----------------------------------
                                  0
```

**Example 6-5 Scoring a Single Text Expression**

```
SELECT
  PREDICTION(T_SVM_Clas_sample USING 'Affinity card is great' AS comments)
FROM DUAL;

PREDICTION(T_SVM_CLAS_SAMPLEUSING'AFFINITYCARDISGREAT'ASCOMMENTS)
-----------------------------------------------------------------
                                                                1
```

# 6.3 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.

## 6.3.1 Cluster Details

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.

**Example 6-6 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;

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>
```

## 6.3.2 Feature Details

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 model named `svd_sh_sample` to the data selected from `svd_sh_sample_build_num`.

**Example 6-7 Feature Details**

```
SELECT FEATURE_DETAILS(svd_sh_sample, 1, 3 USING *) proj1det
  FROM svd_sh_sample_build_num
  WHERE CUST_ID = 101501;

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>
```

## 6.3.3 Prediction Details

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`.

**Example 6-8 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;

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.

**Example 6-9 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;

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.

**Example 6-10 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;

  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>
```

## 6.3.4 GROUPING Hint

The functions used for data mining consist of SQL functions such as `PREDICTION*`, `CLUSTER*`, `FEATURE*`, and `ORA_DM_*`. The `GROUPING` hint is an optional hint which applies to data mining 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 data mining technique 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 <mining attribute list> <right paren>
```

The syntax for only the `PREDICTION` function is given but it is applicable to any data mining technique where `PREDICTION`, `CLUSTERING`, and `FEATURE_EXTRACTION` scoring functions occur.

**Example 6-11 Example**

```
SELECT PREDICTION(/*+ GROUPING */my_model USING *) pred FROM <input
table>;
```

**Related Topics**

- *Oracle Database SQL Language Reference*

# 6.4 Real-Time Scoring

Oracle Data Mining SQL functions enable prediction, clustering, and feature extraction analysis to be easily integrated into live production and operational systems. Because mining results are returned within SQL queries, mining 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.

**Example 6-12 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;

CUST_CARD_PROB
--------------
        .72764
```

# 6.5 Dynamic Scoring

The Data Mining SQL functions operate in two modes: by applying a pre-defined 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 pre-defined model extends the application of basic embedded data mining 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 6-13 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;

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>
```

# 6.6 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 6-1 Sample Cost Matrix**

| ACTUAL_TARGET_VALUE | PREDICTED_TARGET_VALUE | COST |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 2 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Example 6-14 Sample Queries With Costs**

The table `nbmodel_costs` contains the cost matrix described in Table 6-1.

```
SELECT * from nbmodel_costs;
```

```
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;

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 6-1), "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;

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;

C        CNT    AVG_AGE
- ---------- ----------
```

```
F          73          39
M         577          44
```

**Related Topics**

- Example 1-1

# 6.7 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 mining technique-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 6-2 APPLY Output Table**

| Mining Technique | Output Columns |
|---|---|
| classification | CASE_ID<br>PREDICTION<br>PROBABILITY |
| regression | CASE_ID<br>PREDICTION |
| anomaly detection | CASE_ID<br>PREDICTION<br>PROBABILITY |
| clustering | CASE_ID<br>CLUSTER_ID<br>PROBABILITY |
| feature extraction | CASE_ID<br>FEATURE_ID<br>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).

### Example 6-15 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;

   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*

7

# Mining Unstructured Text

Explains how to use Oracle Data Mining to mine unstructured text.



## 7.1 About Unstructured Text

Data mining 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.

## 7.2 About Text Mining and Oracle Text

Understand what is text mining and oracle text.

Text mining is the process of applying data mining 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 a 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 Data Mining uses Oracle Text utilities and term weighting strategies to transform the text for mining. Oracle Data Mining 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*

# 7.3 Data Preparation for Text Features

The model details view for text features is `DM$VX`*model_name*.

The text feature view `DM$VXmodel_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 7-1 Text Feature View for Extracted Text Features**

| Column Name | Description |
|---|---|
| PARTITION_NAME | A partition in a partitioned model to retrieve details |
| COLUMN_NAME | Name of the identifier column |
| TOKEN | Text token which is usually a word or stemmed word |
| DOCUMENT_FREQUENCY | A measure of token frequency in the entire training set |

# 7.4 Creating a Model that Includes Text Mining

Learn how to create a model that includes text mining.

Oracle Data Mining supports unstructured text within columns of `VARCHAR2`, `CHAR`, `CLOB`, `BLOB`, and `BFILE`, as described in the following table:

**Table 7-2 Column Data Types That May Contain Unstructured Text**

| Data Type | Description |
|---|---|
| `BFILE` and `BLOB` | Oracle Data Mining 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` | Oracle Data Mining interprets `CLOB` as text. |
| `CHAR` | Oracle Data Mining interprets `CHAR` as categorical by default. You can identify columns of `CHAR` as text when you create the model. |
| `VARCHAR2` | Oracle Data Mining interprets `VARCHAR2` with data length > 4000 as text.<br>Oracle Data Mining 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 data mining.

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 7-3 Model Settings for Text**

| Setting Name | Data Type | Setting Value | Description |
|---|---|---|---|
| `ODMS_TEXT_POLICY_NAME` | `VARCHAR2(4000)` | Name of an Oracle Text policy object created with `CTX_DDL.CREATE_POLICY` | Affects how individual tokens are extracted from unstructured text. See "Creating a Text Policy". |
| `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`.<br>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 7-3".
3. Specify which columns must be treated as text and, optionally, provide text transformation instructions for individual attributes.
4. Pass the model settings and text transformation instructions to `DBMS_DATA_MINING.CREATE_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).

**Related Topics**

- Specifying Model Settings
  Understand how to configure data mining models at build time.
- Creating 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.
- Configuring a Text Attribute
  Learn how to identify a column as a text attribute and provide transformation instructions for any text attribute.

- Embedding Transformations in a Model

# 7.5 Creating 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 Data Mining 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 7-4 CTX_DDL.CREATE_POLICY Procedure Parameters**

| Parameter Name | Description |
|---|---|
| `policy_name` | Name of the new policy object. Oracle Text policies and text indexes share the same namespace. |
| `filter` | Specifies how the documents must be converted to plain text for indexing. Examples are: `CHARSET_FILTER` for character sets and `NULL_FILTER` for plain text, HTML and XML.<br>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.<br>For `section_group` values, see "Section Group Types" in *Oracle Text Reference*.<br>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).<br>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.<br>The system-supplied stoplist is used by default.<br>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.<br>See "`BASIC_WORDLIST`" in *Oracle Text Reference*. |

**Related Topics**

- *Oracle Text Reference*

# 7.6 Configuring a Text Attribute

Learn how to identify a column as a text attribute and provide transformation instructions for any text attribute.

As shown in Table 7-2, you can identify columns of `CHAR`, shorter `VARCHAR2` (<=4000), `BFILE`, and `BLOB` as text attributes. If `CHAR` and shorter `VARCHAR2` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` processes them as categorical attributes. If `BFILE` and `BLOB` columns are not explicitly identified as unstructured text, then `CREATE_MODEL` returns an error.

To identify a column as a text attribute, supply the keyword `TEXT` in an **Attribute specification**. The attribute specification is a field (`attribute_spec`) in a transformation record (`transform_rec`). Transformation records are components of transformation lists (`xform_list`) that can be passed to `CREATE_MODEL`.

> **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 7-5 Attribute-Specific Text Transformation Instructions**

| Subsetting Name | Description | Example |
|---|---|---|
| `BIGRAM` | A sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words.<br>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 Data Mining supports synonyms. The following is an optional parameter:<br><`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`*)<br>(`TOKEN_TYPE:`*`SYNONYM[NAMES]`*) |

**Table 7-5 (Cont.) Attribute-Specific Text Transformation Instructions**

| Subsetting Name | Description | Example |
|---|---|---|
| TOKEN_TYPE | The following values are supported:<br>NORMAL (the default)<br>STEM<br>THEME<br>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 7-1 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

- Embedding Transformations in a Model

- Specifying Transformation Instructions for an Attribute
  Learn what is a transformation instruction for an attribute and learn about the fields in a transformation record.
- *Oracle Database PL/SQL Packages and Types Reference*
- `ALL_MINING_MODEL_ATTRIBUTES`

# 8
# Administrative Tasks for Oracle Data Mining

Explains how to perform administrative tasks related to Oracle Data Mining.



## 8.1 Installing and Configuring a Database for Data Mining

You can install and configure a database for Oracle Data Mining for SQL by following the listed steps.



### 8.1.1 About Installation

Oracle Data Mining components associated with Oracle Database are included with the database license. This includes Oracle Database Enterprise Edition and Oracle Database Standard Edition 2. Install the 19.9 bundle patch if you want to use Oracle Database Standard Edition 2.

To install Oracle Database, follow the installation instructions for your platform. Choose a Data Warehousing configuration during the installation.

Oracle Data Miner, the graphical user interface to Oracle Data Mining, is an extension to Oracle SQL Developer. Instructions for downloading SQL Developer and installing the Data Miner repository are available on the Oracle Technology Network.

To perform data mining activities, you must be able to log on to the Oracle database, and your user ID must have the database privileges described in Example 8-7.

**Related Topics**

- `Oracle Data Miner`

> **See Also:**
>
> **Install and Upgrade** page of the Oracle Database online documentation library for your platform-specific installation instructions: `Oracle Database 19c Release`

### 8.1.2 Database Tuning Considerations for Data Mining

Standard administrative practices can be followed to manage workload on the system when data mining activities are running.

DBAs managing production databases that support Oracle Data Mining must follow standard administrative practices as described in *Oracle Database Administrator's Guide*.

Building data mining models and batch scoring of mining 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 data mining. The correct sizing of Program Global Area (PGA) memory is very important for model building, complex queries, and batch scoring. From a data mining perspective, the System Global Area (SGA) is generally less of a concern. However, the SGA must be sized to accommodate real-time scoring, which loads models into the shared cursor in the SGA. In most cases, you can configure the database to manage memory automatically. To do so, specify the total maximum memory size in the tuning parameter `MEMORY_TARGET`. With automatic memory management, Oracle Database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands.

Most data mining algorithms can take advantage of parallel execution when it is enabled in the database. Parameters in `INIT.ORA` control the behavior of parallel execution.

**Related Topics**

- *Oracle Database Administrator's Guide*
- Scoring and Deployment
  Explains the scoring and deployment features of Oracle Data Mining.
- *Oracle Database Administrator's Guide*
- Part I Database Performance Fundamentals
- Tuning Database Memory
- *Oracle Database VLDB and Partitioning Guide*

## 8.2 Upgrading or Downgrading Oracle Data Mining

Understand how to upgrade and downgrade Oracle Data Mining.

- Pre-Upgrade Steps
- Upgrading Oracle Data Mining
- Post Upgrade Steps
- Downgrading Oracle Data Mining

## 8.2.1 Pre-Upgrade Steps

Before upgrading, you must drop any data mining models that were created in Java and any mining activities that were created in Oracle Data Miner Classic (the earlier version of Oracle Data Miner).

> **Caution:**
>
> In Oracle Database 12*c*, Oracle Data Mining does not support a Java API, and Oracle Data Miner Classic cannot run against Oracle Database 12*c* .

### 8.2.1.1 Dropping Models Created in Java

If your 10*g* or 11*g* database contains models created in Java, use the `DBMS_DATA_MINING.DROP_MODEL` routine to drop the models before upgrading the database.

### 8.2.1.2 Dropping Mining Activities Created in Oracle Data Miner Classic

If your database contains mining activities from Oracle Data Miner Classic, delete the mining activities and drop the repository before upgrading the database. Follow these steps:

1. Use the Data Miner Classic user interface to delete the mining activities.
2. In SQL*Plus or SQL Developer, drop these tables:

   ```
   DM4J$ACTIVITIES
   DM4J$RESULTS
   DM4J$TRANSFORMS
   ```

   and these views:

   ```
   DM4J$MODEL_RESULTS_V
   DM4J$RESULTS_STATE_V
   ```

There must be no tables or views with the prefix `DM4J$` in any schema in the database after you complete these steps.

## 8.2.2 Upgrading Oracle Data Mining

Learn how to upgrade Oracle Data Mining.

After you complete the "Pre-Upgrade Steps", all models and mining metadata are fully integrated with the Oracle Database upgrade process  whether you are upgrading from 11*g* or from 10*g* 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 mining functionality introduced in the new release.

To upgrade a database, you can use Database Upgrade Assistant (DBUA) or you can perform a manual upgrade using export/import utilities.

**Related Topics**

- Pre-Upgrade Steps
- *Oracle Database Upgrade Guide*

## 8.2.2.1 Using Database Upgrade Assistant to Upgrade Oracle Data Mining

Oracle Database Upgrade Assistant provides a graphical user interface that guides you interactively through the upgrade process.

On Windows platforms, follow these steps to start the Upgrade Assistant:

1. Go to the Windows **Start** menu and choose the Oracle home directory.
2. Choose the **Configuration and Migration Tools** menu.
3. Launch the **Upgrade Assistant**.

On Linux platforms, run the `DBUA` utility to upgrade Oracle Database.

### 8.2.2.1.1 Upgrading from Release 10*g*

In Oracle Data Mining 10g, data mining metadata and PL/SQL packages are stored in the `DMSYS` schema. In Oracle Data Mining 11*g* and 12*c*, `DMSYS` no longer exists; data mining metadata objects are stored in `SYS`.

When Oracle Database 10*g* is upgraded to 12*c*, all data mining metadata objects and PL/SQL packages are migrated from `DMSYS` to `SYS`. The `DMSYS` schema and its associated objects are removed after a successful migration. When `DMSYS` is removed, the `SYS.DBA_REGISTRY` view no longer lists Oracle Data Mining as a component.

After upgrading to Oracle Database 12*c*, you can no longer switch to the Data Mining Scoring Engine (`DMSE`). The Scoring Engine does not exist in Oracle Database 11*g* or 12*c*.

### 8.2.2.1.2 Upgrading from Release 11*g*

If you upgrade Oracle Database 11*g* to Oracle Database 12*c*, and the database was previously upgraded from Oracle Database 10*g*, then the`DMSYS` schema may still be present. If the upgrade process detects `DMSYS`, it displays a warning message and drops `DMSYS` during the upgrade.

## 8.2.2.2 Using Export/Import to Upgrade Data Mining Models

If required, you can you can use a less automated approach to upgrading data mining models. You can export the models created in a previous version of Oracle Database and import them into an instance of Oracle Database 12*c*.

> **Caution:**
>
> Do not import data mining models that were created in Java. They are not supported in Oracle Database 12*c*.

### 8.2.2.2.1 Export/Import Release 10*g* Data Mining Models

Follow the instructions for exporting and importing Data Mining models.

To export models from an instance of Oracle Database 10*g* to a dump file, follow the instructions in "Exporting and Importing Mining Models". Before importing the models from the dump file, run the `DMEIDMSYS` script to create the `DMSYS` schema in Oracle Database 12*c*.

```
SQL>CONNECT / as sysdba;
SQL>@ORACLE_HOME\RDBMS\admin\dmeidmsys.sql
SQL>EXIT;
```

> **Note:**
>
> The `TEMP` tablespace must already exist in the Oracle Database 12*g* database. The `DMEIDMSYS` script uses the `TEMP` and `SYSAUX` tablespaces to create the `DMSYS` schema.

To import the dump file into the Oracle Database 12*c* database:

```
%ORACLE_HOME\bin\impdp system\<password>
       dumpfile=<dumpfile_name>
       directory=<directory_name>
       logfile=<logfile_name> .....
SQL>CONNECT / as sysdba;
SQL>EXECUTE dmp_sys.upgrade_models();
SQL>ALTER SYSTEM FLUSH SHARED_POOL;
SQL>ALTER SYSTEM FLUSH BUFFER_CACHE;
SQL>EXIT;
```

The `upgrade_models` script migrates all data mining metadata objects and PL/SQL packages from `DMSYS` to `SYS` and then drops `DMSYS` before upgrading the models.

#### ALTER SYSTEM Statement

You can flush the Database Smart Flash Cache by issuing an `ALTER SYSTEM FLUSH FLASH_CACHE` statement. Flushing the Database Smart Flash Cache can be useful if you need to measure the performance of rewritten queries or a suite of queries from identical starting points.

#### Related Topics

- Exporting and Importing Mining Models
  You can export machine learning models to move models to a different Oracle Database instance, such as from a development database to a production database.

### 8.2.2.2.2 Export/Import Release 11*g* Data Mining Models

To export models from an instance of Oracle Database 11*g* to a dump file, follow the instructions in Exporting and Importing Mining Models.

> **Caution:**
>
> Do not import data mining models that were created in Java. They are not supported in Oracle Database 12*c*.

To import the dump file into the Oracle Database 12*c* database:

```
%ORACLE_HOME\bin\impdp system\<password>
       dumpfile=<dumpfile_name>
       directory=<directory_name>
       logfile=<logfile_name> .....
SQL>CONNECT / as sysdba;
SQL>EXECUTE dmp_sys.upgrade_models();
SQL>ALTER SYSTEM flush shared_pool;
SQL>ALTER SYSTEM flush buffer_cache;
SQL>EXIT;
```

**ALTER SYSTEM Statement**

You can flush the Database Smart Flash Cache by issuing an `ALTER SYSTEM FLUSH FLASH_CACHE` statement. Flushing the Database Smart Flash Cache can be useful if you need to measure the performance of rewritten queries or a suite of queries from identical starting points.

## 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 mining 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 `12.1`.

> **Note:**
>
> The `CREATE MINING MODEL` privilege must be granted to Data Mining user accounts that are used to create mining models.

**Related Topics**

- Creating a Data Mining User
  Explains how to create a Data Mining user.
- Controlling Access to Mining Models and Data
  Understand how to create a Data Mining user and grant necessary privileges.

## 8.2.4 Downgrading Oracle Data Mining

Before downgrading the Oracle Database 12*c* database back to the previous version, ensure that no Singular Value Decomposition models or Expectation Maximization models are present. These algorithms are only available in Oracle Database 12*c*. Use

the DBMS_DATA_MINING.DROP_MODEL routine to drop these models before downgrading. If you do not do this, the database downgrade process terminates.

Issue the following SQL statement in SYS to verify the downgrade:

```
SQL>SELECT o.name FROM sys.model$ m, sys.obj$ o
                  WHERE m.obj#=o.obj# AND m.version=2;
```

# 8.3 Exporting and Importing Mining Models

You can export machine learning models to move models to a different Oracle Database instance, such as from a development database to a production database.

The DBMS_DATA_MINING package includes procedures for migrating machine learning models between database instances.

EXPORT_MODEL exports a single model or list of models to a dump file so it can be imported, queried, and scored in a separate Oracle Machine Learning database instance.

IMPORT_MODEL takes the dump file and creates the model in the destination database.

EXPORT_SERMODEL exports a single model to a serialized BLOB so it can be imported and scored in a separate Oracle Data Mining database instance or to OML Services.

IMPORT_SERMODEL takes the serialized BLOB and creates the model in the destination database.

- About Exporting Models
- About Oracle Data Pump
- Options for Exporting and Importing Mining Models
- Directory Objects for EXPORT_MODEL and IMPORT_MODEL
- Using EXPORT_MODEL and IMPORT_MODEL
- EXPORT and IMPORT Serialized Models
- Importing From PMML

**Related Topics**

- EXPORT_MODEL
- IMPORT_MODEL
- EXPORT_SERMODEL
- IMPORT_SERMODEL

## 8.3.1 About Exporting Models

As a result of building models, each model has a set of model detail views that provide information about the model, such as model statistics for evaluation. The user can query these model detail views. With serialized models, only the model data and metadata required for scoring are available in the serialized model. This is more compact and transfers faster to the destination environment than dump files produced by the EXPORT_MODEL procedure.

To retain complete model details, use the DMBS_DATA_MINING.EXPORT_MODEL procedure and the DBMS_DATA_MINING.IMPORT_MODEL procedure. Serialized model export only works with

models that produce scores. Specifically, it doesn't support Attribute Importance, Association Rules, Exponential Smoothing, or O-Cluster (although O-Cluster does allow scoring). Use `EXPORT_MODEL` to export these models and scenarios when full model details are needed.

**Related Topics**

- EXPORT_MODEL Procedure
- IMPORT_MODEL Procedure

## 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 data mining models, to a dump file set. The Data Pump import utility retrieves the schema objects, including the model tables and metadata, from the dump file set and restores them in the target database.

`expdp` and `impdp` cannot be used to export/import individual data mining 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 Mining Models

Lists options for exporting and importing mining models.

Options for exporting and importing mining models are described in the following table.

**Table 8-1 Export and Import Options for Oracle Data Mining 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 data mining models in the database are included. |
| Export or import a schema | Use `expdp` to export a schema and `impdp` to import a schema. All data mining 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 data mining model, all data mining models, or data mining models that match specific criteria.<br>To import models, you must have the `CREATE TABLE`, `CREATE VIEW`, and `CREATE MINING MODEL` privileges. |

**Table 8-1 (Cont.) Export and Import Options for Oracle Data Mining for SQL**

| Task | Description |
|---|---|
| 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`.<br>To create a private database link, you must have the `CREATE DATABASE LINK` system privilege. To create a public database link, you must have the `CREATE PUBLIC DATABASE LINK` system privilege. Also, you must have the `CREATE SESSION` system privilege on the remote Oracle Database. Oracle Net must be installed on both the local and remote Oracle Databases. |
| Serialized model export and import | Starting from Oracle Database 18c, the serialized model format was introduced as a lightweight approach to support scoring. The `DBMS_DATA_MINING.EXPORT_SERMODEL` procedure exports a single model to a serialized `BLOB` so it can be imported and scored in a separate Oracle Data Mining database instance. `DBMS_DATA_MINING.IMPORT_SERMODEL` takes the serialized `BLOB` and creates the model in the target database. |

**Related Topics**

- IMPORT_MODEL Procedure
- EXPORT_MODEL Procedure
- *Oracle Database SQL Language Reference*

## 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 data mining models, you must have write access to the directory object and to the file system directory that it represents. To import data mining 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 `dmdir`. 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/dmuser`, the command is:

```
CREATE OR REPLACE DIRECTORY dmdir AS '/home/dmuser';
```

The following SQL command gives user `dmuser` both read and write access to `dmuser_dir`.

```
GRANT READ,WRITE ON DIRECTORY dmdir TO dmuser;
```

**Related Topics**

- *Oracle Database SQL Language Reference*

## 8.3.5 Using 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 `dmdir` shown in Example 8-1 and two schemas, `dm1` and `dm2`. Both schemas have data mining privileges. `dm1` has two models. `dm2` has one model.

```
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
DM2        SVD_SH_SAMPLE        FEATURE_EXTRACTION
SINGULAR_VALUE_DECOMP
```

**Example 8-1 Creating the Directory Object**

```
-- connect as system user
CREATE OR REPLACE DIRECTORY OMLDIR AS '/home/oracle';
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM1;
GRANT READ, WRITE ON DIRECTORY OMLDIR TO DM2;
SELECT * FROM all_directories WHERE directory_name = 'DMDIR';
```

The output is as follows:

```
OWNER      DIRECTORY_NAME             DIRECTORY_PATH
---------- --------------------------
----------------------------------------
SYS        DMDIR                      /home/dmuser
```

**Example 8-2 Exporting All Models From DM1**

```
-- connect as DM1
BEGIN
  dbms_data_mining.export_model (
                  filename =>   'all_DM1',
                  directory =>  'DMDIR');
END;
/
```

A log file and a dump file are created in `/home/dmuser`, the physical directory associated with `DMDIR`. 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 => 'DMDIR');
END;
/
SELECT model_name FROM user_mining_models;
```

The output is as follows:

```
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 => 'DMDIR',
                  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 =>'DMDIR',
            model_filter => 'name in (''DT_SH_CLAS_SAMPLE'')');
-- one_model01.dmp and dm1_exp_37.log are created in /home/dmuser

-- Export Decision Tree models
EXECUTE dbms_data_mining.export_model(
            filename => 'algo_models',
            directory => 'DMDIR',
            model_filter => 'ALGORITHM_NAME IN (''DECISION_TREE'')');
-- algo_model01.dmp and dm1_exp_410.log are created in /home/dmuser

-- Export clustering models
EXECUTE dbms_data_mining.export_model(
            filename =>'func_models',
            directory => 'DMDIR',
            model_filter => 'FUNCTION_NAME = ''CLUSTERING''');
-- func_model01.dmp and dm1_exp_513.log are created in /home/dmuser
```

**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 Importing 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://dmg.org/). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Data Mining 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 Controlling Access to Mining Models and Data

Understand how to create a Data Mining user and grant necessary privileges.

- Creating a Data Mining User
- System Privileges for Data Mining
- Object Privileges for Mining Models

## 8.4.1 Creating a Data Mining User

Explains how to create a Data Mining user.

A Data Mining user is a database user account that has privileges for performing data mining activities. Example 8-6 shows how to create a database user. Example 8-7 shows how to assign data mining privileges to the user.

> **Note:**
>
> To create a user for the Data Mining sample programs, you must run two configuration scripts as described in "The Data Mining Sample Programs".

**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 `dmuser`, type these commands. Specify a password of your choosing.

```
CREATE USER dmuser IDENTIFIED BY password
       DEFAULT TABLESPACE USERS
       TEMPORARY TABLESPACE TEMP
       QUOTA UNLIMITED ON USERS;
Commit;
```

   The `USERS` and `TEMP` tablespace are included in the pre-configured database that Oracle ships with the database media. `USERS` is used mostly by demo users; it is appropriate for running the sample programs described in "The Data Mining Sample Programs". `TEMP` is the temporary tablespace that is shared by most database users.

   > **Note:**
   >
   > Tablespaces for Data Mining users must be assigned according to standard DBA practices, depending on system load and system resources.

3. To login as `dmuser`, type the following.

```
CONNECT dmuser
Enter password: password
```

**Related Topics**

- The Data Mining Sample Programs
  Describes the data mining sample programs that ship with Oracle Database.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the complete syntax of the `CREATE USER` statement

#### 8.4.1.1 Granting Privileges for Data Mining

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 data mining privileges to the `dmuser` account. Some of these privileges are not required for all mining activities, however it is prudent to grant them all as a group.

Additional system and object privileges are required for enabling or restricting specific mining activities.

**Example 8-7 Privileges Required for Data Mining**

```
GRANT CREATE MINING MODEL TO dmuser;
GRANT CREATE SESSION TO dmuser;
GRANT CREATE TABLE TO dmuser;
GRANT CREATE VIEW TO dmuser;
GRANT EXECUTE ON CTXSYS.CTX_DDL TO dmuser;
```

`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 dmuser;
```

### 8.4.2 System Privileges for Oracle Data Mining 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 data mining 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 data mining models.

**Table 8-2 System Privileges for Oracle Data Mining for SQL**

| System Privilege | Allows you to.... |
|---|---|
| `CREATE MINING MODEL` | Create data mining models in your own schema. |
| `CREATE ANY MINING MODEL` | Create data mining models in any schema. |
| `ALTER ANY MINING MODEL` | Change the name or cost matrix of any data mining model in any schema. |
| `DROP ANY MINING MODEL` | Drop any data mining model in any schema. |
| `SELECT ANY MINING MODEL` | Apply a data mining model in any schema, also view model details in any schema. |
| `COMMENT ANY MINING MODEL` | Add a comment to any data mining model in any schema. |
| `AUDIT_ADMIN` role | Generate an audit trail for any data mining model in any schema. (See *Oracle Database Security Guide* for details.) |

**Example 8-8 Grant System Privileges for Oracle Data Mining 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 data mining activities in the `oml_user` schema.

```
REVOKE SELECT ANY MINING MODEL FROM oml_user;
```

**Related Topics**

- Adding a Comment to a Mining Model
- *Oracle Database Security Guide*

## 8.4.3 Object Privileges for Mining Models

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 mining models.

**Table 8-3 Object Privileges for Mining Models**

| Object Privilege | Allows you to.... |
|---|---|
| `ALTER MINING MODEL` | Change the name or cost matrix of the specified mining model object. |
| `SELECT MINING MODEL` | Apply the specified mining model object and view its model details. |

**Example 8-9 Grant Object Privileges on Mining Models**

The following statements allow `dmuser` to apply the model `testmodel` to the `sales` table, specifying different cost matrixes with each apply. The user `dmuser` can also rename the model `testmodel`. The `testmodel` model and `sales` table are in the `sh` schema, not in the `dmuser` schema.

```
GRANT SELECT ON MINING MODEL sh.testmodel TO dmuser;
GRANT ALTER ON MINING MODEL sh.testmodel TO dmuser;
GRANT SELECT ON sh.sales TO dmuser;
```

The following statement prevents `dmuser` from renaming or changing the cost matrix of `testmodel`. However, `dmuser` can still apply `testmodel` to the `sales` table.

```
REVOKE ALTER ON MINING MODEL sh.testmodel FROM dmuser;
```

# 8.5 Auditing and Adding Comments to Mining Models

Mining model objects support SQL `COMMENT` and `AUDIT` statements.

## 8.5.1 Adding a Comment to a Mining Model

Comments can be used to associate descriptive information with a database object. You can associate a comment with a mining 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;

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 8-2
- *Oracle Database SQL Language Reference* for details about SQL `COMMENT` statements

## 8.5.2 Auditing Mining Models

The Oracle Database auditing system is a powerful, highly configurable tool for tracking operations on schema objects in a production environment. The auditing system can be used to track operations on data mining models.

**Note:**

To audit mining 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 Data Mining Events" in *Oracle Database Security Guide* for details about auditing mining models
- "Monitoring Database Activity with Auditing" in *Oracle Database Security Guide* for a comprehensive discussion of unified auditing in Oracle Database
- "About the Unified Auditing Migration Process for Oracle Database" in *Oracle Database Upgrade Guide* for information about migrating to unified auditing
- *Oracle Database Upgrade Guide*

# A
# The Data Mining Sample Programs

Describes the data mining sample programs that ship with Oracle Database.



## A.1 About the Data Mining Sample Programs

You can learn a great deal about the Oracle Data Mining application programming interface (API) from the data mining sample programs. The programs illustrate typical approaches to data preparation, algorithm selection, algorithm tuning, testing, and scoring.

The programs are easy to use. They include extensive inline comments to help you understand the code. They delete all temporary objects on exit; you can run the programs repeatedly without setup or cleanup.

The data mining sample programs are installed with Oracle Database Examples in the demo directory under Oracle Home. The demo directory contains sample programs that illustrate many features of Oracle Database. You can locate the data mining files by doing a directory listing of `dm*.sql`. The following example shows this directory listing on a Linux system.

Note that the directory listing in the following example includes one file, `dmhpdemo.sql`, that is *not* a data mining program.

**Example A-1 Directory Listing of the Data Mining Sample Programs**

```
> cd $ORACLE_HOME/rdbms/demo
> ls dm*.sql
dmaidemo.sql      dmkmdemo.sql    dmsvddemo.sql
dmardemo.sql      dmnbdemo.sql    dmsvodem.sql
dmdtdemo.sql      dmnmdemo.sql    dmsvrdem.sql
dmdtxvlddemo.sql  dmocdemo.sql    dmtxtnmf.sql
dmemdemo.sql      dmsh.sql        dmtxtsvm.sql
dmglcdem.sql      dmshgrants.sql
dmglrdem.sql      dmstardemo.sql
dmhpdemo.sql      dmsvcdem.sql
```

The data mining sample programs create a set of mining models in the user's schema. After executing the programs, you can list the models with a query like the one in the following example.

**Example A-2 Models Created by the Sample Programs**

```
SELECT mining_function, algorithm, model_name FROM user_mining_models
    ORDER BY mining_function;

MINING_FUNCTION               ALGORITHM                      MODEL_NAME
----------------------------- ------------------------------ --------------------
```

| | | |
|---|---|---|
| ASSOCIATION_RULES | APRIORI_ASSOCIATION_RULES | AR_SH_SAMPLE |
| CLASSIFICATION | GENERALIZED_LINEAR_MODEL | GLMC_SH_CLAS_SAMPLE |
| CLASSIFICATION | SUPPORT_VECTOR_MACHINES | T_SVM_CLAS_SAMPLE |
| CLASSIFICATION | SUPPORT_VECTOR_MACHINES | SVMC_SH_CLAS_SAMPLE |
| CLASSIFICATION | SUPPORT_VECTOR_MACHINES | SVMO_SH_CLAS_SAMPLE |
| CLASSIFICATION | NAIVE_BAYES | NB_SH_CLAS_SAMPLE |
| CLASSIFICATION | DECISION_TREE | DT_SH_CLAS_SAMPLE |
| CLUSTERING | EXPECTATION_MAXIMIZATION | EM_SH_CLUS_SAMPLE |
| CLUSTERING | O_CLUSTER | OC_SH_CLUS_SAMPLE |
| CLUSTERING | KMEANS | KM_SH_CLUS_SAMPLE |
| CLUSTERING | KMEANS | DM_STAR_CLUSTER |
| FEATURE_EXTRACTION | SINGULAR_VALUE_DECOMP | SVD_SH_SAMPLE |
| FEATURE_EXTRACTION | NONNEGATIVE_MATRIX_FACTOR | NMF_SH_SAMPLE |
| FEATURE_EXTRACTION | NONNEGATIVE_MATRIX_FACTOR | T_NMF_SAMPLE |
| REGRESSION | SUPPORT_VECTOR_MACHINES | SVMR_SH_REGR_SAMPLE |
| REGRESSION | GENERALIZED_LINEAR_MODEL | GLMR_SH_REGR_SAMPLE |

# A.2 Installing the Data Mining Sample Programs

Learn how to install Data Mining sample programs.

The data mining sample programs require:

- Oracle Database Enterprise Edition with the Advanced Analytics option
- Oracle Database sample schemas
- Oracle Database Examples
- A data mining user account
- Execution of `dmshgrants.sql` by a system administrator
- Execution of `dmsh.sql` by the data mining user

Follow these steps to install the data mining sample programs:

1. Install or obtain access to Oracle Database 19*c* Enterprise Edition or Oracle Standard Edition 2 (SE2). To install the Database, see the installation instructions for your platform at `Oracle Database 19c Release`.
2. Ensure that the sample schemas are installed in the database. The sample schemas are installed by default with Oracle Database. See *Oracle Database Sample Schemas*for details about the sample schemas.
3. Verify that Oracle Database Examples has been installed with the database, or install it locally. Oracle Database Examples loads the Database sample programs into the `rdbms/demo` directory under Oracle home. See *Oracle Database Examples Installation Guide* for installation instructions.
4. Verify that a data mining user account has been created, or create it yourself if you have administrative privileges. See "Creating a Data Mining User".
5. Ask your system administrator to run `dmshgrants.sql`, or run it yourself if you have administrative privileges. `dmshgrants` grants the privileges that are required for running the sample programs. These include `SELECT` access to tables in the `SH` schema as described in "The Data Mining Sample Data" and the system privileges listed in the following table.

   Pass the name of the data mining user to `dmshgrants`.

```
SQL> CONNECT sys / as sysdba
Enter password: sys_password
Connected.
SQL> @ $ORACLE_HOME/rdbms/demo/dmshgrants dmuser
```

**Table A-1 System Privileges Granted by dmshgrants.sql to the Data Mining User**

| Privilege | Allows the data mining 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 data mining models |
| EXECUTE ON ctxsys.ctx_ddl | execute procedures in the ctxsys.ctx_ddl PL/SQL package; required for text mining |

6. Connect to the database as the data mining user and run dmsh.sql. This script creates views of the sample data in the schema of the data mining user.

```
SQL> CONNECT dmuser
Enter password: dmuser_password
Connected.
SQL> @ $ORACLE_HOME/rdbms/demo/dmsh
```

**Related Topics**

- *Oracle Database Sample Schemas*
- *Oracle Database Examples Installation Guide*
- Creating a Data Mining User
  Explains how to create a Data Mining user.

# A.3 The Data Mining Sample Data

The data used by the sample data mining programs is based on these tables in the SH schema:

```
SH.CUSTOMERS
SH.SALES
SH.PRODUCTS
SH.SUPPLEMENTARY_DEMOGRAPHICS
SH.COUNTRIES
```

The dmshgrants script grants SELECT access to the tables in SH. The dmsh.sql script creates views of the SH tables in the schema of the data mining user. The views are described in the following table:

**Table A-2 The Data Mining Sample Data**

| 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 |

**Table A-2 (Cont.) The Data Mining Sample Data**

| View Name | Description |
|---|---|
| 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 program creates its own transactional data.

# Index

## E



## F



## G



## I



## K



## L



## M

## N



## O



## P



## R



## S

## T



## U



## W



## X