Oracle® Data Mining

Application Developer's Guide

11*g* Release 2 (11.2)

E12218-07

July 2011

ORACLE®

Oracle Data Mining Application Developer's Guide, 11*g* Release 2 (11.2)

E12218-07

Primary Author: Kathy L. Taylor

Contents

5 Building a Model

6 Scoring and Deployment

7 The Data Mining Java API

Index

Preface

This manual describes the programmatic interfaces to Oracle Data Mining. You can use the PL/SQL and Java interfaces to create data mining applications or add data mining features to existing applications. You can use the data mining SQL operators in applications or in ad hoc queries.

> **Note:** The Oracle Data Mining Java API is deprecated in this release.
>
> Oracle recommends that you not use deprecated features in new applications. Support for deprecated features is for backward compatibility only

This manual should be used along with the demo applications and the related reference documentation. (See "Related Documentation" on page ii-vi.)

The preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

Audience

This manual is intended for database programmers who are familiar with Oracle Data Mining.

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

Related Documentation

The documentation set for Oracle Data Mining is part of the Oracle Database 11*g* Release 2 (11.2) Online Documentation Library. The Oracle Data Mining documentation set consists of the following documents:

- *Oracle Data Mining Concepts*
- *Oracle Data Mining Java API Reference* (javadoc)
- *Oracle Data Mining Administrator's Guide*

> **Note:** Information to assist you in installing and using the Data Mining demo programs is provided in *Oracle Data Mining Administrator's Guide*.

- *Oracle Database PL/SQL Packages and Types Reference* (`DBMS_DATA_MINING`, `DBMS_DATA_MINING_TRANSFORM`, and `DBMS_PREDICTIVE_ANALYTICS`)
- *Oracle Database SQL Language Reference* (Data Mining functions)

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.

What's New in the Oracle Data Mining APIs?

This section describes new features in the Oracle Data Mining APIs. It includes the following sections:

- Oracle Data Mining 11g Release 2 (11.2.0.3) API New Features
- Oracle Data Mining 11g Release 2 (11.2.0.1) API New Features

> **See Also:** *Oracle Data Mining Concepts* for additional information about new features and for information about features that were new or deprecated in Oracle Data Mining 11*g* Release 1

Oracle Data Mining 11*g* Release 2 (11.2.0.3) API New Features

The Oracle Data Mining Java API is deprecated in this release.

> **Note:** Oracle recommends that you not use deprecated features in new applications. Support for deprecated features is for backward compatibility only

Oracle Data Mining 11*g* Release 2 (11.2.0.1) API New Features

This section lists the changes that have been introduced in the Oracle Data Mining 11.2 PL/SQL API:

- **Support for Native Transactional Data with Association Rules**

  In Oracle Data Mining 11*g* Release 2 (11.2), you can build association rules models without first transforming the transactional data.

  > **See:** "Market Basket Data" on page 3-11

- **SVM Class Weights Specified with CLAS_WEIGHTS_TABLE_NAME**

  Previously SVM class weights were specified in the priors table (`CLAS_PRIORS_TABLE_NAME` setting). Now SVM class weights and GLM class weights are both specified in a class weights table (`CLAS_WEIGHTS_TABLE_NAME` setting)

  > **See:** `DBMS_DATA_MINING` setting `CLAS_WEIGHTS_TABLE_NAME` in *Oracle Database PL/SQL Packages and Types Reference*

- **FORCE argument to DROP_MODEL**

You can now force a drop model operation even if a serious system error has interrupted the model build process.

> **See:** `DBMS_DATA_MINING.DROP_MODEL` procedure in *Oracle Database PL/SQL Packages and Types Reference*

- **GET_MODEL_DETAILS_SVM has new REVERSE_COEF parameter**

  To preserve model transparency, the `GET_MODEL_DETAILS` functions automatically reverse the transformations generated by ADP during the model build. You can obtain the transformed attribute coefficients used internally by an SVM model by setting the new `reverse_coef` parameter to 1. This causes the coefficients and bias to be returned with the normalization shifts and scales applied by ADP.

  > **See:** `DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM` in *Oracle Database PL/SQL Packages and Types Reference*

1
Data Mining API Use Cases

This chapter shows how you can perform complex data mining operations by using simple SQL queries and PL/SQL procedures. The examples show techniques for obtaining valuable information about your customers. This kind of information is needed to answer business questions, such as: How should we position a product? When should we introduce a new product? What pricing and sales promotion strategies should we use?

> **Note:** The base interfaces to Oracle Data Mining are PL/SQL packages and SQL functions. Most of the examples throughout this manual use PL/SQL and SQL code.

See Also:

- Chapter 6, "Scoring and Deployment"
- *Oracle Database SQL Language Reference* for syntax of the SQL Data Mining functions
- *Oracle Database PL/SQL Packages and Types Reference* for syntax of the PL/SQL API

This chapter contains the following topics:

- Analyze Customer Demographics and Buying Patterns
- Evaluate the Success of a Marketing Campaign
- Use Predictive Analytics to Create a Customer Profile

Analyze Customer Demographics and Buying Patterns

The code fragments in this section show how you can apply data mining models to learn more about your customers.

These examples use models that were previously created. They exist in Oracle Database as database objects. For the sake of simplicity, we can assume that you created the models, that they exist within your schema, and that you are executing the SQL statements that apply them. You could have used the PL/SQL API or Oracle Data Miner to create the models.

See Also: Chapter 5, "Building a Model"

The primary method for applying data mining models is by executing specialized SQL functions. There exists a separate family of SQL functions for prediction, clustering, and feature extraction.

See Also: Chapter 6, "Scoring and Deployment"

Note: The examples in the first two sections of this chapter illustrate SQL syntax for different use cases, but the code uses hypothetical object names. For example, a table might be named *my_table*.

Segment Customer Base and Predict Attrition

The query in Example 1–1 divides customers into segments and predicts customer attrition.

The segments are created by a clustering model named clus_model. The model uses heuristics to divide the customer database into groups that have similar characteristics.The prediction is generated by a classification model called svmC_model.

The model predicts the attrition probability for each customer in each segment. The data is returned by segment, ordered by the overall attrition propensity in the segment.

Example 1–1 Predict Attrition for Customer Segments

```
SELECT count(*) as cnt,
       AVG(PREDICTION_PROBABILITY(svmC_model,
               'attrite' USING *)) as avg_attrite,
       AVG(cust_value_score)
  FROM customers
 GROUP BY CLUSTER_ID(clus_model USING *)
 ORDER BY avg_attrite DESC;
```

The sophisticated analytics within this seemingly simple query let you see your customer base within natural groupings based on similarities. The SQL data mining functions show you where your most loyal and least loyal customers are. This information can help you make decisions about how to sell your product. For example, if most of the customers in one segment have a high probability to attrite, you might take a closer look at the demographics of that segment when considering your marketing strategy.

The CLUSTER_ID function in Example 1–1 applies the mining model clus_model to the customers table. It returns the segment for each customer. The number of segments is determined by the clustering algorithm. Oracle Data Mining supports two clustering algorithms: enhanced *k*-Means and O-Cluster.

The PREDICTION_PROBABILITY function in Example 1–1 applies the mining model svmC_model and returns the probability to attrite for each customer. Oracle Data Mining supports a number of classification algorithms. In this case, the model name implies that Support Vector Machine was chosen as the classifier.

Predict Missing Incomes

The sample query in Example 1–2 returns the ten customers who are most likely to attrite, based on age, gender, annual_income, and zipcode.

In addition, since annual_income is often missing, the PREDICTION function is used to perform missing value imputation for the annual_income attribute. The PREDICTION function applies the regression model, svmR_model, to predict the most likely annual income. The NVL function replaces missing annual income values with the resulting prediction.

Example 1–2 Predict Customer Attrition and Transform Missing Income Values

```
SELECT * FROM (
  SELECT cust_name, cust_contact_info
    FROM customers
   ORDER BY
     PREDICTION_PROBABILITY(tree_model, 'attrite'
       USING age, gender, zipcode,
         NVL(annual_income,
             PREDICTION(svmR_model USING *))
           as annual_income) DESC)
WHERE rownum < 11;
```

Find Anomalies in the Customer Data

These examples use a one-class SVM model to discover atypical customers (outliers), find common demographic characteristics of the most typical customers, and compute the probability that a new or hypothetical customer will be a typical affinity card holder.

Find the Top 10 Outliers

Find the top 10 outliers -- customers that differ the most from the rest of the population. Depending on the application, such atypical customers can be removed from the data.

```
SELECT cust_id FROM (
 SELECT cust_id
 FROM svmo_sh_sample_prepared
  ORDER BY prediction_probability(SVMO_SH_Clas_sample, 0 using *) DESC, 1)
WHERE rownum < 11;
```

Compute the Probability of a New Customer Being a Typical Affinity Card Member

Compute the probability of a new or hypothetical customer being a typical affinity card holder. Normalization of the numerical attributes is performed on-the-fly.

```
WITH age_norm AS (
   SELECT shift, scale FROM SVMO_SH_sample_norm WHERE col = 'AGE'),
     yrs_residence_norm AS (
       SELECT shift, scale FROM svmo_sh_sample_norm WHERE col = 'YRS_RESIDENCE')
       SELECT prediction_probability(SVMO_SH_Clas_sample, 1 using
                             (44 - a.shift)/a.scale AS age,
                             (6 - b.shift)/b.scale 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 age_norm a, yrs_residence_norm b;
```

Find the Demographics of a Typical Affinity Card Member

Find demographic characteristics of the typical affinity card members. These statistics will not be influenced by outliers and are likely to provide a more truthful picture of the population of interest than statistics computed on the entire group of affinity members.

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

Evaluate the Success of a Marketing Campaign

This example uses a classification model to predict who will respond to a marketing campaign for DVDs and why.

1. First predict the responders.

   This statement uses the `PREDICTION` and `PREDICTION_DETAILS` functions to apply the model `campaign_model`.

   ```
   SELECT cust_name,
      PREDICTION(campaign_model USING *)
         AS responder,
      PREDICTION_DETAILS(campaign_model USING *)
         AS reason
   FROM customers;
   ```

2. Combine the predictions with relational data.

   This statement combines the predicted responders with additional information from the `sales` table. In addition to predicting the responders, it shows how much each customer has spent for a period of three months before and after the start of the campaign.

   ```
   SELECT cust_name,
           PREDICTION(campaign_model USING *) AS responder,
           SUM(CASE WHEN purchase_date < 15-Apr-2005 THEN
               purchase_amt ELSE 0 END) AS pre_purch,
           SUM(CASE WHEN purchase_date >= 15-Apr-2005 THEN
               purchase_amt ELSE 0 END) AS post_purch
    FROM customers, sales
    WHERE sales.cust_id = customers.cust_id
      AND purchase_date BETWEEN 15-Jan-2005 AND 14-Jul-2005
    GROUP BY cust_id, PREDICTION(campaign_model USING *);
   ```

3. Combine the predictions and relational data with multi-domain, multi-database data.

   In addition to predicting responders, find out how much each customer has spent on DVDs for a period of three months before and after the start of the campaign.

   ```
   SELECT cust_name,
           PREDICTION(campaign_model USING *) as responder,
           SUM(CASE WHEN purchase_date < 15-Apr-2005 THEN
               purchase_amt ELSE 0 END) AS pre_purch,
           SUM(CASE WHEN purchase_date >= 15-Apr-2005 THEN
               purchase_amt ELSE 0 END) AS post_purch
   ```

```
 FROM customers, sales, products@PRODDB
 WHERE sales.cust_id = customers.cust_id
   AND purchase_date BETWEEN 15-Jan-2005 AND 14-Jul-2005
   AND sales.prod_id = products.prod_id
   AND CONTAINS(prod_description, 'DVD') > 0
 GROUP BY cust_id, PREDICTION(campaign_model USING *);
```

4. Evaluate the effectiveness and significance of the information you have obtained.

   Compare the success rate of predicted responders and non-responders within different regions and across the company. Is the success statistically significant?

```
SELECT responder, cust_region, COUNT(*) AS cnt,
       SUM(post_purch - pre_purch) AS tot_increase,
       AVG(post_purch - pre_purch) AS avg_increase,
       STATS_T_TEST_PAIRED(pre_purch, post_purch) AS significance
FROM (
 SELECT cust_name, cust_region
        PREDICTION(campaign_model USING *) AS responder,
        SUM(CASE WHEN purchase_date < 15-Apr-2005 THEN
            purchase_amt ELSE 0 END) AS pre_purch,
        SUM(CASE WHEN purchase_date >= 15-Apr-2005 THEN
            purchase_amt ELSE 0 END) AS post_purch
 FROM customers, sales, products@PRODDB
 WHERE sales.cust_id = customers.cust_id
   AND purchase_date BETWEEN 15-Jan-2005 AND 14-Jul-2005
   AND sales.prod_id = products.prod_id
   AND CONTAINS(prod_description, 'DVD') > 0
 GROUP BY cust_id, PREDICTION(campaign_model USING *) )
GROUP BY ROLLUP responder, cust_region ORDER BY 4 DESC;
```

Use Predictive Analytics to Create a Customer Profile

Predictive analytics, implemented in the DBMS_PREDICTIVE_ANALYTICS package, supports routines for making predictions, assessing attribute importance, and creating profiles. Example 1–3 shows how you could use predictive analytics to generate a customer profile.

Note: With predictive analytics, you do not need to create a model. The routine dynamically creates and applies a model, which does not persist upon completion.

See Also: *Oracle Data Mining Concepts* for an overview of Oracle predictive analytics

The PROFILE statement in Example 1–3 returns rules that suggest whether or not a customer is likely to use an affinity card. The rules are generated based on two predictors: customer gender and customer occupation. The rules are written as XML to a results table with these columns.

```
Name                        Type
-------------------------    ------------
PROFILE_ID                   NUMBER
RECORD_COUNT                 NUMBER
DESCRIPTION                  XMLTYPE
```

The rule identifier is stored in PROFILE_ID. The number of cases described by the rule is stored in RECORD_COUNT. The XML that describes the rule is stored in the DESCRIPTION column.

Note: This example uses sample data based on the SH schema. This data is used with the Data Mining sample programs. For information on the sample programs, see *Oracle Data Mining Administrator's Guide*.

Example 1–3 Generate a Customer Profile

```
--create a source view
CREATE VIEW cust_gend_occ_view AS
                 SELECT cust_gender, occupation, affinity_card
                 FROM mining_data_apply;

-- describe the source data
DESCRIBE cust_gend_occ_view
 Name                            Null?    Type
 ----------------------------------------------
 CUST_GENDER                     VARCHAR2(1)
 OCCUPATION                      VARCHAR2(21)
 AFFINITY_CARD                   NUMBER(10)

-- find the rules
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.PROFILE(
         DATA_TABLE_NAME    => 'cust_gend_occ_view',
         TARGET_COLUMN_NAME => 'affinity_card',
         RESULT_TABLE_NAME  => 'profile_result');
END;
/

-- PROFILE has created 5 rules
SELECT profile_id from cust_gend_occ_profile_results;

PROFILE_ID
----------
         1
         2
         3
         4
         5

-- display the rules

<SimpleRule id="1" score="1" recordCount="275">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="CUST_GENDER" booleanOperator="isIn">
      <Array type="string">"M"</Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="1" recordCount="146"/>
<ScoreDistribution value="0" recordCount="129"/>
</SimpleRule>
```

```
<SimpleRule id="2" score="0" recordCount="124">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="CUST_GENDER" booleanOperator="isIn">
      <Array type="string">"F"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="0" recordCount="96"/>
<ScoreDistribution value="1" recordCount="28"/>
</SimpleRule>

<SimpleRule id="3" score="0" recordCount="397">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="CUST_GENDER" booleanOperator="isIn">
      <Array type="string">"M"
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Crafts" "Sales" "TechSup" "Transp."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="0" recordCount="289"/>
<ScoreDistribution value="1" recordCount="108"/>
</SimpleRule>

<SimpleRule id="4" score="0" recordCount="316">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="CUST_GENDER" booleanOperator="isIn">
      <Array type="string">"M"
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">
        "?" "Cleric." "Farming" "Handler" "House-s" "Machine" "Other"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="0" recordCount="277"/>
<ScoreDistribution value="1" recordCount="39"/>
</SimpleRule

<SimpleRule id="5" score="0" recordCount="388">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">
        "?" "Cleric." "Crafts" "Farming" "Handler" "House-s" "Machine"
        "Other" "Sales" "TechSup" "Transp."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="CUST_GENDER" booleanOperator="isIn">
      <Array type="string">"F"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
<ScoreDistribution value="0" recordCount="363"/>
```

```
<ScoreDistribution value="1" recordCount="25"/>
</SimpleRule>
```

2

A Tour of the Data Mining APIs

This chapter provides an overview of the PL/SQL, SQL, and Java interfaces to Oracle Data Mining.

> **Note:** The Oracle Data Mining Java API is deprecated in this release.
>
> Oracle recommends that you not use deprecated features in new applications. Support for deprecated features is for backward compatibility only

This chapter contains the following sections:

- Data Mining PL/SQL Packages
- Data Mining Data Dictionary Views
- Data Mining SQL Functions
- Data Mining Java API

Data Mining PL/SQL Packages

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

- **DBMS_DATA_MINING**, the primary interface to Oracle Data Mining
- **DBMS_DATA_MINING_TRANSFORM**, convenience routines for data transformation
- **DBMS_PREDICTIVE_ANALYTICS**, predictive analytics

DBMS_DATA_MINING

The DBMS_DATA_MINING package includes procedures for:

- Creating, dropping, and renaming mining models
- Applying a model to new data
- Describing the model details
- Creating costs and computing test metrics for a classification model
- Exporting and importing models

CREATE_MODEL

The CREATE_MODEL procedure creates a mining model. The attributes, transformations, rules, and other information internal to the model are returned by GET_MODEL_DETAILS functions. You can also obtain information about mining models by querying data dictionary views, as described in "Data Mining Data Dictionary Views" on page 2-3.

APPLY

The APPLY procedure creates a table with specific columns and populates the columns with mining results. The columns of this table vary based on the particular mining function and algorithm.

> **Note:** The Data Mining SQL functions, introduced in Oracle Database 10.2, are now generally the preferred method for applying Data Mining models. See "Data Mining SQL Functions" on page 3.

DBMS_DATA_MINING_TRANSFORM

This package includes routines for transforming the data to make it suitable for mining. Since Oracle Data Mining supports Automatic Data Transformation (ADP), you will not need to use this package unless you want to implement specialized transformations.

You can supplement the ADP-generated transformations with additional transformations that you specify yourself, or you can elect to transform the data yourself instead of using ADP.

The routines in DBMS_DATA_MINING_TRANSFORM are convenience routines to assist you in creating your own transformations. If these routines do not entirely suit your needs, you can write SQL to modify their output, or you can write your own routines.

To specify transformations for a model, pass a transformation list to the DBMS_DATA_MINING.CREATE_MODEL procedure. You can use the STACK procedures in DBMS_DATA_MINING_TRANSFORM to build the transformation list.

Oracle Data Mining embeds automatic transformations and transformations you pass to CREATE_MODEL in the model. The embedded transformations are automatically applied to the data when the model is applied. You do not need to separately transform the test or scoring data.

See Also:

- DBMS_DATA_MINING_TRANSFORM in *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Data Mining Concepts* for information about automatic and embedded data transformation

DBMS_PREDICTIVE_ANALYTICS

This package includes routines for predictive analytics, an automated form of data mining. With predictive analytics, you do not need to be aware of model building or scoring. All mining activities are handled internally by the predictive analytics procedure.

Predictive analytics routines prepare the data, build a model, score the model, and return the results of model scoring. Before exiting, they delete the model and supporting objects.

Oracle predictive analytics supports 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.

> **See Also:** DBMS_PREDICTIVE_ANALYTICS in *Oracle Database PL/SQL Packages and Types Reference*

Data Mining Data Dictionary Views

You can obtain information about mining models from the data dictionary. The data dictionary views for Oracle Data Mining are available for ALL_, USER_, and DBA_ access.

The Data Mining data dictionary views are summarized as follows:

- **ALL_MINING_MODELS** returns information about the mining models to which you have access.

  See "Mining Model Schema Objects" on page 5-8.
- **ALL_MINING_MODEL_ATTRIBUTES** returns information about the attributes of the mining models to which you have access.

  See "About Attributes" on page 3-3.
- **ALL_MINING_MODEL_SETTINGS** returns information about the settings for the mining models to which you have access.

  See "Model Settings" on page 5-2.

Data Mining SQL Functions

The built-in SQL functions for Data Mining implement scoring operations for models that have already been created in the database. They 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, especially in the case of single row scoring.
- Scoring results are pipelined, enabling the rows to be returned iteratively as they are produced. Pipelining is an optimization that greatly improves performance. For information about pipelining, see *Oracle Database PL/SQL Language Reference*.

> **Note:** SQL functions are built into Oracle Database and are available for use within SQL statements. SQL functions should not be confused with functions defined in PL/SQL packages.

When applied to a given row of scoring data, classification and regression models provide the best predicted value for the target and the associated probability of that value occurring. The SQL functions for prediction are described in Table 2–1.

Table 2–1 SQL Functions for Prediction

Function	Description
PREDICTION	Returns the best prediction for the target
PREDICTION_BOUNDS	(GLM only) Returns the upper and lower bounds of the interval wherein the values (linear regression) or probabilities (logistic regression) will lie
PREDICTION_COST	Returns a measure of the cost of incorrect predictions

Table 2–1 (Cont.) SQL Functions for Prediction

Function	Description
`PREDICTION_DETAILS`	Returns the rules of a Decision Tree model
`PREDICTION_PROBABILITY`	Returns the probability of a given prediction
`PREDICTION_SET`	Returns the results of a classification model, including the predictions and associated probabilities for each case

Applying a cluster model to a given row of scoring data returns the cluster ID and the probability of that row's membership in the cluster. The SQL functions for clustering are described in Table 2–2.

Table 2–2 SQL Functions for Clustering

Function	Description
`CLUSTER_ID`	Returns the ID 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

Applying a feature extraction model involves the mapping of features (sets of attributes) to columns in the scoring data set. The SQL functions for feature extraction are described in Table 2–3.

Table 2–3 SQL Functions for Feature Extraction

Function	Description
`FEATURE_ID`	Returns the ID of the feature with the highest coefficient value
`FEATURE_SET`	Returns a list of objects containing all possible features along with the associated coefficients
`FEATURE_VALUE`	Returns the value of a given feature

See Also: Chapter 6, "Scoring and Deployment"

Data Mining Java API

The Oracle Data Mining Java API is an Oracle implementation of the JDM standard (JSR-73) Java API. It is a thin API developed using the rich in-database functionality of Oracle Data Mining.

Note: The Oracle Data Mining Java is deprecated in this release.

Oracle recommends that you not use deprecated features in new applications. Support for deprecated features is for backward compatibility only

The Oracle Data Mining Java API implements Oracle specific extensions to provide all the data mining features available in the database. All extensions are designed to be compliant with the JDM standards extension framework. All the mining functions and algorithms available in the database are exposed through the Oracle Data Mining Java API.

Oracle Database 10.2.0.1 introduced the JDM 1.0 standard compliant API that replaced the old Oracle proprietary Java API in the previous releases. Database 10.2.0.2 patch-set release extended the JDM standards support by implementing Oracle Data Mining Java API compatible with JDM 1.1.

In this release, the Oracle Data Mining Java API continues to be compatible with the JDM 1.1 and provides new data mining functionality in the database server as Oracle extensions. In this release new Oracle features include automatic and embedded data preparation, generalized linear models, transformation sequencing and task dependency specifications.

The JDM Standard

JDM is an industry standard Java API for data mining developed under the Java Community Process (JCP). It defines Java interfaces that vendors can implement for their Data Mining Engine (DME). It includes interfaces supporting mining functions such as classification, regression, clustering, attribute importance and association along with specific mining algorithms such as naïve bayes, support vector machines, decision trees, feed forward neural networks, and *k*-means.

An overview of the Java packages defined by the standards is listed in Table 2–4. For more details, refer to the Java documentation published with the standard at `http://www.jcp.org`. In the **Go to JSR** box, type in `73`.

Table 2–4 JDM Standard Java Packages

Package	Description
javax.datamining	Defines objects supporting all JDM subpackages.
javax.datamining.base	Defines objects supporting many top-level mining objects. Introduced to avoid cyclic package dependencies.
javax.datamining.resource	Defines objects that support connecting to the Data Mining ENgine and executing tasks.
javax.datamining.data	Defines objects supporting logical and physical data, model signature, taxonomy, category set and the generic super class category matrix.
javax.datamining.statistics	Defines objects supporting attribute statistics.
javax.datamining.rules	Defines objects supporting rules and their predicate components.
javax.datamining.task	Defines objects supporting tasks for build, compute statistics, import, and export. Task has an optional subpackage for apply since apply is used mainly for supervised and clustering functions.
javax.datamining.association	Defines objects supporting the build settings and model for association.
javax.datamining.clustering	Defines objects supporting the build settings and model for clustering.
javax.datamining.attributeimportance	Defines objects supporting the build settings and model for attribute importance.
javax.datamining.supervised	Defines objects supporting the build settings and models for supervised learning functions, specifically classification and regression, with corresponding optional packages. It also includes a common test task for the classification and regression functions.
javax.datamining.algorithm	Defines objects supporting the settings that are specific to algorithms. The algorithm package has optional sub packages for different algorithms.
javax.datamining.modeldetail	Defines objects supporting details of various model representation. Model Details has optional sub packages for different model details.

Oracle Extensions to JDM

Oracle extensions are defined to support the functionality that is not part of the JDM standards. This section gives an overview of these extensions.

> **See Also:** *Oracle Data Mining Java API Reference* (javadoc).

Oracle extensions have the following major additional features:

- Feature Extraction function with the Non-negative Matrix Factorization (NMF) algorithm
- Generalized Linear Model algorithm for regression and classification functions
- Oracle-proprietary clustering algorithm, Orthogonal Partitioning Clustering (O-CLuster)
- Oracle-proprietary classification algorithm, Adaptive Bayes Network (ABN) (deprecated)
- Automated and embedded data transformations
- Predictive analytics tasks

An overview of the Oracle extensions higher-level Java packages is provided in Table 2–5.

Table 2–5 Oracle Extensions Higher-Level Packages

Package	Description
oracle.dmt.jdm.featureextraction	Defines the objects related to the feature extraction function. Feature extraction supports the scoring operation.
oracle.dmt.jdm.algorithm.nmf	Defines the objects related to the Non-negative Matrix Factorization (NMF) algorithm.
oracle.dmt.jdm.algorithm.glm oracle.dmt.jdm.modeldetail.glm	Defines the objects related to the Generalized Linear Model (GLM) algorithm.
oracle.dmt.jdm.algorithm.ocluster	Defines the objects related to the Orthogonal Clustering (O-Cluster) algorithm.
oracle.dmt.jdm.algorithm.abn	Defines the objects related to the Adaptive Bayes Network (ABN) algorithm (deprecated).
oracle.dmt.jdm.transform	Defines the objects related to the transformations.

Principal Objects in the Oracle Data Mining Java API

In JDM, named objects are objects that can be saved using the save method in the Connection. All the named objects are inherited from the `javax.datamining.MiningObject` interface. You can choose to persist the named objects either permanently (persistent objects) or only for the lifetime of the Connection object (transient objects).

Table 2–6 lists the JDM named objects supported by Oracle Data Mining.

Table 2–6 JDM Named Objects Supported by Oracle Data Mining

Persistent Objects	Transient Objects	Unsupported Objects
Model	Apply Settings	Logical Data
Build Settings	Physical Dataset	Taxonomy

Table 2–6 (Cont.) JDM Named Objects Supported by Oracle Data Mining

Persistent Objects	Transient Objects	Unsupported Objects
Task		
Cost Matrix		
Test Metrics		
Transformation sequence		

Physical Data Set

Physical data sets refer to the data to be used as input to data mining operations. PhysicalDataSet objects reference specific data identified by a URI. Oracle Data Mining supports a table or view in the same database as a valid physical dataset URI. Syntax of the physical dataset URI is as follows:

Data URI Syntax:

[schemaName.] tableName/viewName

The PhysicalDataSet object can support multiple data representations. Oracle Data Mining supports two types of data representation: single-record case and wide data. (See Chapter 3 for details.) The Oracle Data Mining implementation requires users to specify the case ID column in the physical dataset.

A PhysicalDataSet object is a transient object in the Oracle Data Mining Java API. It is stored in the Connection as an in-memory object.

Build Settings

A BuildSettings object captures the high-level specification input for building a model. The API specifies mining functions: classification, regression, attribute importance, association, clustering, and feature extraction.

Build settings allow a user to specify the type of result desired without having to specify a particular algorithm. Although a build settings object allows for the specification of an algorithm and its settings, if the algorithm settings are omitted, the DME selects an algorithm based on the build settings and possibly characteristics of the data.

Build settings may also be validated for correct parameters using the verify method.

A BuildSettings object is persistent; it is stored as a table with a user-specified name in the user's schema. This settings table is interoperable with the PL/SQL API. Oracle recommends that you not modify the build settings table manually.

Task

The Execute method in the Connection object is used to start an execution of a mining task. Typically, mining operations are done using tables with millions of records, so the execution of operations such as a model build can take a long time.

JDM supports asynchronous execution of mining tasks using `DBMS_SCHEDULER` in the database. Each mining task is stored as a `DBMS_SCHEDULER` job object in the user's schema. When the user saves the task object, it creates a job object and sets the object to be in the `DISABLED` state. When the user executes a task, it enables the job to start execution.

To monitor tasks that are executed asynchronously, the Execute method returns a `javax.datamining.ExecutionHandle` object. It provides methods such as `waitForCompletion` and `getStatus` to retrieve the status details of the task.

Model

A Model object is the result of applying an algorithm to data as specified in a BuildSettings object.

Models can be used in several operations. They can be:

- Inspected, for example to examine the rules produced from a decision tree or association
- Tested for accuracy
- Applied to data for scoring
- Exported to an external representation such as native format or PMML
- Imported for use in the DME

When a model is applied to data, it is submitted to the DME for interpretation. A Model object references its `BuildSettings` object as well as the `Task` that created it. as.

Test Metrics

A TestMetrics object is the result of testing a supervised model with test data. Based on the type of mining function, different test metrics are computed. For classification models, accuracy, confusion-matrix, lift, and receiver-operating characteristic can be computed to access the model. Similarly for regression models, R-squared and RMS errors can be computed.

Apply Settings

An ApplySettings object allows users to tailor the results of an apply task. It contains a set of ordered items. Output can consist of:

- Data (key attributes) to be passed through to the output from the input dataset.
- Values computed from the apply itself: score, probability, and in the case of decision trees, rule identifiers.
- Multi-class categories for its associated probabilities. For example, in a classification model with target favoriteColor, users could select the specific colors to receive the probability that a given color is favorite.

Each mining function class defines a method to construct a default apply settings object. This simplifies the programmer's effort if only standard output is desired. For example, typical output for a classification apply would include the top prediction and its probability.

Transformation Sequence

A TransformationSequence object represents the sequence of transformations that are to be performed as part of a mining operation. For example, a Support Vector Machine model build involves outlier handling and normalization transformations. In addition to this, there can be new derived attribute creation and business transformations, and so on. Typically these transformations are reused for other model builds and hence applications can save the transformation sequence as a named object in the API.

Transformation sequences can be used either to perform transformations as a separate task or embed them to the modeling process by specifying it as one of the input objects for model building.

See Also: Chapter 7 for more details about the Java API

3

Creating the Case Table

This chapter explains how data is interpreted by Oracle Data Mining. It describes the requirements for a data mining case table, and it explains the notion of data attributes and model attributes. Data transformations are discussed in *Oracle Data Mining Concepts*.

This chapter contains the following sections:

Requirements

The data that you wish to mine must be 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 be identified by a unique **case ID**. The table or view itself is 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 Example 3–1 are selected from SH.CUSTOMERS.

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
```

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. See "Market Basket Data" on page 3-11.

Column Data Types

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 and 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 accepts the following column data types:

```
VARCHAR2, CHAR
NUMBER, FLOAT
DM_NESTED_CATEGORICALS
DM_NESTED_NUMERICALS
```

See *Oracle Data Mining Concepts* for information about converting the data type if necessary.

The nested types are described in "Nested Data" on page 3-7. The case ID column cannot have a nested type.

Data Sets for Data Mining

You need two case tables to build a classification or regression model. One set of rows is used for building (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 might select 60% of the rows for building the model and 40% for testing the model.

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

Most data mining models can be applied to separate data. The data to which you apply the model is called the **apply data** or **scoring data**. 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 build data but `NUMBER` in the scoring data, Oracle Data Mining will effectively apply 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. Mining models that contain embedded transformations are known as **supermodels**.

Automatic and embedded data transformations are discussed in *Oracle Data Mining Concepts*.

About Attributes

Attributes are the items of data 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. A table of employee data might contain attributes such as job title, date of hire, salary, age, gender, and so on.

Data Attributes and Model Attributes

Data attributes are columns in the data sets 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`, might be an attribute used by an algorithm to build a model. Internally, the model attribute `SIZE` would 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, would not correspond to a model attribute. The data attribute would be `SALES_PROD`, but each product with its corresponding sales figure (each row in the nested column) would be a model attribute.

Transformations also cause a discrepancy between data attributes and model attributes. For example, a transformation could apply a calculation to two data attributes and store the result in a new attribute. The new attribute would be 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.

See Also:

- "Nested Data" on page 3-7
- *Oracle Data Mining Concepts* for information on transformations

Target Attribute

The **target** of a supervised model is a special kind of attribute. The target column in the build data contains the historical values used to build (train) the model. The target column in the test data contains the historical values to which the predictions are compared. The target column in the scoring data holds the results when the model is applied.

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

You can query the *_MINING_MODEL_ATTRIBUTES view to find the target for a given model, as shown in Example 3–2.

Numericals and Categoricals

Model attributes are either **numerical** or **categorical**. Data attributes, which are columns in a case table, have Oracle data types.

Oracle Data Mining interprets NUMBER, FLOAT, and DM_NESTED_NUMERICALS as numerical, and CHAR, VARCHAR2, and DM_NESTED_CATEGORICALS as categorical. There is one exception: If the target of a classification model is NUMBER or FLOAT, it will be interpreted as categorical.

Numerical attributes can theoretically have an infinite number of values. The values have an implicit order, and the differences between them are also ordered.

Categorical attributes have values that belong to 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. The term **multi-class** is used to describe models when the categorical target has more than two values. For example, a target of clothing sizes could have the values small, medium, or large.

The target of a classification model is categorical. The target of a regression model is numerical. The target of an attribute importance model is either categorical or numerical.

Model Signature

The **model signature** is the set of data attributes used to build a model. Some or all of the attributes in the signature should be present for scoring. If some columns are not present, they are disregarded. If columns with the same names but different data types are present, the model attempts to convert the data type.

The model signature does not necessarily include all the columns in the build data. Algorithm-specific criteria may cause the model to ignore certain columns. Other columns may 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.

ALL_MINING_MODEL_ATTRIBUTES

The columns in the model signature, plus the target (if the model is supervised), are listed in the data dictionary view, ALL/USER/DBA_MINING_MODEL_ATTRIBUTES. When used with the ALL prefix, this view returns the signature and target for all mining models accessible to the current user. When used with the USER prefix, it returns the model signature and target for all the mining models in the user's schema. The DBA prefix is only available to DBAs.

The columns in the ALL_MINING_MODEL_ATTRIBUTES view are described as follows. Details are in Table 3–1.

```
SQL> describe all_mining_model_attributes
 Name                            Null?    Type
 ----------------------------------------------------------
 OWNER                           NOT NULL VARCHAR2(30)
 MODEL_NAME                      NOT NULL VARCHAR2(30)
 ATTRIBUTE_NAME                NOT NULL VARCHAR2(30)
 ATTRIBUTE_TYPE                         VARCHAR2(11)
 DATA_TYPE                              VARCHAR2(12)
 DATA_LENGTH                            NUMBER
 DATA_PRECISION                         NUMBER
 DATA_SCALE                             NUMBER
 USAGE_TYPE                             VARCHAR2(8)
 TARGET                                 VARCHAR2(3)
```

Table 3–1 ALL_MINING_MODEL_ATTRIBUTES

Column	Description
OWNER	Owner of the mining model.
MODEL_NAME	Name of the mining model.
ATTRIBUTE_NAME	Name of the data attribute (column).
ATTRIBUTE_TYPE	Type of the model attribute derived by the model from the data attribute. The attribute type can be either numerical or categorical. This information is only meaningful if there is a one-to-one mapping between the data attribute and the model attribute. If the data attribute has undergone transformations that change the way it is used by the model, then the ATTRIBUTE_TYPE may not be relevant.
DATA_TYPE	The Oracle data type of the data attribute (column): NUMBER FLOAT CHAR VARCHAR2 NESTED TABLE If the value is NESTED TABLE, the data type is either: DM_NESTED_NUMERICALS or DM_NESTED_CATEGORICALS If the data type is NESTED TABLE, you can determine whether it is DM_NESTED_NUMERICALS or DM_NESTED_CATEGORICALS from the ATTRIBUTE_TYPE column. See "Nested Data" on page 3-7 for details.
DATA_LENGTH	Length of the data type
DATA_PRECISION	Precision of a fixed point number, which is the total number of significant decimal digits, is represented as p in the data type NUMBER (p,s).
DATA_SCALE	Scale of a fixed point number. Scale, which is the number of digits from the decimal to the least significant digit, is represented as *s* in the data type NUMBER (p,*s*).
USAGE_TYPE	Indicates that the attribute was used to construct the model. Some attributes may be eliminated by transformations or algorithmic processing. The *_MINING_MODEL_ATTRIBUTES view only lists the data attributes used by the model (model signature), therefore the value of USAGE_TYPE is always ACTIVE.
TARGET	Whether or not the attribute is a target. The value can be either YES (the attribute is a target) or NO (the attribute is not a target). If the attribute is a target, and it has undergone transformations for manipulation by the algorithm, the description in ALL_MINING_MODEL_ATTRIBUTES reflects the target attribute's characteristics after reverse transformations have been applied.

The query in Example 3–2 returns information about the data attributes of the model T_SVM_CLAS_SAMPLE, an SVM model generated by one of the Data Mining sample programs. The query returns the name and data type of each of the data attributes in the model signature, whether the attribute is used as a numerical or as a categorical, and whether or not the attribute is a target.

Example 3–2 ALL_MINING_MODEL_ATTRIBUTES

```
SQL> select model_name, attribute_name, attribute_type, data_type, target
             from user_mining_model_attributes
             where model_name = 'T_SVM_CLAS_SAMPLE';

MODEL_NAME          ATTRIBUTE_NAME         ATTRIBUTE_TYPE  DATA_TYPE     TARGET
------------------- ---------------------  --------------- -----------   ------
T_SVM_CLAS_SAMPLE   COMMENTS               NUMERICAL       NESTED TABLE  NO
T_SVM_CLAS_SAMPLE   AGE                    NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   CUST_MARITAL_STATUS    CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   COUNTRY_NAME           CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   CUST_INCOME_LEVEL      CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   EDUCATION              CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   OCCUPATION             CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   HOUSEHOLD_SIZE         CATEGORICAL     VARCHAR2      NO
T_SVM_CLAS_SAMPLE   YRS_RESIDENCE          NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   BULK_PACK_DISKETTES    NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   FLAT_PANEL_MONITOR     NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   HOME_THEATER_PACKAGE   NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   BOOKKEEPING_APPLICATION NUMERICAL      NUMBER        NO
T_SVM_CLAS_SAMPLE   PRINTER_SUPPLIES       NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   Y_BOX_GAMES            NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   OS_DOC_SET_KANJI       NUMERICAL       NUMBER        NO
T_SVM_CLAS_SAMPLE   CUST_GENDER            CATEGORICAL     CHAR          NO
T_SVM_CLAS_SAMPLE   AFFINITY_CARD          NUMERICAL       NUMBER        YES
```

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 also have a subcolumn_name component as shown in Example 3–3.

Example 3–3 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
```

Model Details

Model details reveal information about model attributes and their treatment by the algorithm. There is a separate GET_MODEL_DETAILS routine for each algorithm.

Transformation and reverse transformation expressions are associated with model attributes. The transformations are applied to the model for algorithmic processing. The reverse transformations are applied for model details. The information returned to the user by GET_MODEL_DETAILS is expressed in the form of the original data attributes, or as close to it as possible.

Reverse transformations are also applied to the target when a supervised model is scored. Reverse transformations support **model transparency**. Transparency is discussed in *Oracle Data Mining Concepts*.

Example 3–4 shows the definition of the GET_MODEL_DETAILS function for an Attribute Importance model. The PIPELINED keyword instructs Oracle Database to return the rows as single values instead of returning all the rows as a single value.

Example 3–4 Model Details for an Attribute Importance Model

The syntax of the GET_MODEL_DETAILS function for Attribute Importance models is shown as follows.

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_AI (
             model_name             VARCHAR2)
RETURN DM_RANKED_ATTRIBUTES PIPELINED;
```

The function returns DM_RANKED_ATTRIBUTES, a virtual table. The columns are the model details. There is one row for each model attribute in the specified model. The columns are described as follows.

```
(attribute_name          VARCHAR2(4000),
 attribute_subname       VARCHAR2(4000),
 importance_value        NUMBER,
 rank                    NUMBER(38))
```

Nested Data

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 might 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.

> **See Also:** Sample code for converting to a nested table in "Example: Creating a Nested Column for Mining" on page 3-10.

The algorithms that support nested data are listed in Table 3–2

Table 3–2 Oracle Data Mining Algorithms that Support Nested Data

Algorithm	Mining Function
Apriori	association rules
GLM	classification and regression
k-Means	clustering
MDL	attribute importance
Naive Bayes	classification
NMF	feature extraction
SVM	classification, regression, and anomaly detection

Nested Object 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. You can learn more about collection types in *Oracle Database Object-Relational Developer's Guide*.

Oracle Data Mining supports two nested object types: one for numerical attributes, the other for categorical attributes.

DM_NESTED_NUMERICALS

The `DM_NESTED_NUMERICALS` object type is a nested table of numerical attributes. Each row is a single `DM_NESTED_NUMERICAL`.

The nested numerical attributes (rows) are described as follows.

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

The collection of numerical attributes (table) is described as follows.

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

DM_NESTED_CATEGORICALS

The `DM_NESTED_CATEGORICALS` object type is a nested table of categorical attributes. Each row is a single `DM_NESTED_CATEGORICAL`.

The nested categorical attributes (rows) are described as follows.

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

The collection of categorical attributes (table) is described as follows.

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

Example: Transforming Transactional Data for Mining

Example 3–5 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 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 3–6 shows how this data could be transformed for mining. The case ID column would be `PRODUCT`. `SALES_PER_REGION`, a nested column of type `DM_NESTED_NUMERICALS`, would be a data attribute. This table is suitable for mining, because the information for each case is stored in a single row.

Example 3–6 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)
```

```
.
.
```

Oracle Data Mining treats each nested row as a separate model attribute, as shown in Example 3–7. (Note that the presentation in this example is conceptual only. The data is not actually pivoted before being processed.)

Example 3–7 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
.
.
```

Example: Creating a Nested Column for Mining

Example 3–8 shows how to define a nested column for data mining. This example uses transactional market basket data.

Example 3–8 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.

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

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.

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

This query returns two rows from the transformed data.

```
SQL> 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', 2),
           DM_NESTED_NUMERICAL('External 8X CD-ROM', 1),
           DM_NESTED_NUMERICAL('SIMM- 16MB PCMCIAII card', 1))
```

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 might assert with 65% confidence that 80% of the people who buy peanut butter 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 or not 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:

- See *Oracle Database PL/SQL Packages and Types Reference* for information on the `ODMS_ITEM_ID_COLUMN_NAME` and `ODMS_ITEM_VALUE_COLUMN_NAME` settings
- "Missing Data" on page 3-11.

Missing Data

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 may have a quantity, while the items that are not represented are sparse (with a known quantity of zero).

How Oracle Data Mining Interprets Missing Data

Oracle Data Mining interprets missing data as follows:

- **Missing** — 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.

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.

Sparsity in a Sales Table

A sales table contains point-of-sale data for a group of products, sold in several stores to different customers over a period of time. A particular customer will only have bought some of the products. Those products that a customer did not buy will 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 would 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 would be represented as nested data for mining.

Since missing values in a nested column will always indicate sparsity, you should make sure that this interpretation is appropriate for the data that you wish to mine. For example, when trying to mine a multi-record case data set containing users' movie ratings of a large movie database, the missing ratings would be unknown (missing at random), but Oracle Data Mining would treat the data as sparse and infer a rating of zero for the missing value.

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 may be 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.

How Oracle Data Mining Treats Missing Data

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 Table 3–3.

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	SVM, NMF, *k*-Means, GLM	NB, MDL, DT, OC	Apriori
NUMERICAL missing at random	Oracle Data Mining replaces missing numerical values with the mean.	The algorithm handles missing values naturally as missing at random.	The algorithm interprets all missing data as sparse.
CATEGORICAL missing at random	Oracle Data Mining replaces missing categorical values with the mode.	The algorithm handles missing values naturally as missing random.	The algorithm interprets all missing data as sparse.
NUMERICAL sparse	Oracle Data Mining replaces sparse numerical data with zeros.	DT and O-Cluster do not support nested data, and therefore do not support sparse data. NB and MDL replace sparse numerical data with zeros.	The algorithm handles sparse data.
CATEGORICAL sparse	Oracle Data Mining replaces sparse categorical data with zero vectors.	DT and O-Cluster do not support nested data, and therefore do not support sparse data. NB and MDL replace sparse categorical data with the special value `DM$SPARSE`.	The algorithm handles sparse data.

Attribute Transformation and Missing Data Treatment

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 would interpret 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 will not perform missing value treatment if there is a specified value. See *Oracle Database SQL Language Reference*

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.

4

Preparing Text for Mining

Oracle Data Mining supports the mining of data sets that have one or more text columns. These columns must undergo a special preprocessing step whereby text tokens known as **terms** are extracted and stored in a nested column. The transformed text can then be used as any other attribute in the building, testing, and scoring of models. Any algorithm that supports nested data can be used for text mining. (See Table 3–2 in Chapter 3.)

This chapter explains how to use PL/SQL to prepare a column of text for mining.

> **Note:** Oracle Data Mining includes sample programs that illustrate text transformation and text mining using the PL/SQL API. Refer to *Oracle Data Mining Administrator's Guide* for information about the Oracle Data Mining sample programs.

> **See Also:** *Oracle Data Mining Concepts* for information about text mining

This chapter contains the following sections.

Oracle Text Routines for Term Extraction

Oracle Data Mining uses specialized Oracle Text routines to preprocess text data. Oracle Text is a technology within the database for building text querying and classification applications.

Oracle Text provides the following facilities for the Oracle Data Mining term extraction process:

- `SVM_CLASSIFIER`, defined in the `CTX_DLL` Oracle Text PL/SQL package, specifies an index preference for Oracle Data Mining term extraction.
- The `CTXSYS.DRVODM` Oracle Text PL/SQL package defines the table functions, `FEATURE_PREP` and `FEATURE_EXPLAIN`, which generate intermediate and final tables of text terms for Oracle Data Mining.

In PL/SQL, the term extraction process requires the use of these Oracle Text facilities. (See "Using Text Transformation" on page 7-25.)

See Also: *Oracle Text Application Developer's Guide* and *Oracle Text Reference* for information on Oracle Text

Note: The Oracle Text facilities for Oracle Data Mining are documented in this chapter. They are not documented in the Oracle Text manuals.

Note on Terminology: Text terms are also known as **features**. In text mining, a feature is a word or group of words extracted from a text document. Features (terms) are the fundamental unit of text that can be manipulated and analyzed.

The feature extraction mining function (NMF algorithm in Oracle Data Mining) and text mining transformation both perform a kind of feature extraction.

- A feature extraction model creates features that represent fundamental qualities of multiple attributes. The model operates on the features instead of the original attributes.
- Text transformation changes a text document into a collection of features, each one representing a fundamental characteristic of the document. The model operates on the text features instead of the original document.

Term Extraction in the Sample Programs

A good place to start in learning the term extraction process is with the sample programs. The following sample programs contain term extraction code for text mining:

- `dmsh.sql` — Prepares the build, test, and scoring data for the sample programs, including the text mining programs. `dmsh.sql` creates views for data mining and tables and indexes for text mining.
- `dmtxtfe.sql` — Using the indexed text column created by `dmsh.sql`, creates a nested column suitable for text mining.

  The `dmtxtfe.sql` program is a sample term extractor. It contains extensive comments that explain the code in a step-by-step fashion. You can expand this program into a complete term extraction solution by adding index creation and the preparation of test and scoring data (as in `dmsh.sql`).

Two sample PL/SQL programs use the data prepared by `dmsh.sql` to mine text.

- `dmtxtnmf.sql` creates and applies a model that uses Non-Negative Matrix Factorization.
- `dmtxtsvm.sql` creates and applies a model that uses SVM classification.

Both these programs mine a table of customer data that includes a nested column of text data called `COMMENTS`. The `COMMENTS` column has been pre-processed by `dmsh.sql`. The models created by these programs are referenced in the following example from a Linux system.

```
-- Run the programs to create the models
SQL> @ $ORACLE_HOME/rdbms/demo/dmtxtnmf.sql
SQL> @ $ORACLE_HOME/rdbms/demo/dmtxtsvm.sql
-- List the models created by the programs
SQL> SELECT model_name, mining_function, algorithm FROM user_mining_models;

MODEL_NAME                MINING_FUNCTION              ALGORITHM
------------------------  ---------------------------  ------------------------
T_SVM_CLAS_SAMPLE         CLASSIFICATION               SUPPORT_VECTOR_MACHINES
T_NMF_SAMPLE              FEATURE_EXTRACTION           NONNEGATIVE_MATRIX_FACTOR
```

See Also: *Oracle Data Mining Administrator's Guide*. This manual provides instructions for obtaining and running the sample programs. It includes information about the build, training, and scoring data used by these programs.

From Unstructured Data to Structured Data

The pre-processing steps for text mining create nested table columns of type DM_NESTED_NUMERICALS from columns of type VARCHAR2 or CLOB. Each row of the nested table specifies an attribute name and a value. The DM_NESTED_NUMERICALS type defines the following columns.

```
attribute_name     VARCHAR2(4000)
value               NUMBER)
```

The term extraction process treats the text in each row of the original table as a separate document. Each document is transformed to a set of terms that have a numeric value and a text label. Within the nested table column, the attribute_name column holds the text and the value column holds the numeric value of the term, which is derived using the term frequency in the document and in the document collection (other rows).

For example, the following query returns various attributes of customer 102998, including a text column of comments. The text column has not been transformed.

```
SQL> select cust_id, cust_gender, cust_income_level, affinity_card, comments
             from mining_build_text
             where cust_id = 102998;

CUST_ID C  CUST_INCOME_LEVEL     AFFINITY_CARD COMMENTS
------- -- --------------------  ------------- --------------------------------
102998  M  J: 190,000 - 249,999  1             I wanted to write you to let you
                                               know that I've purchased several
                                               items at your store recently and
                                               have been very satisfied with my
                                               purchases. Keep up the good work.
```

The following query returns the same attributes of customer 102998, but the text in the comments column has been transformed. The query extracts the ATTRIBUTE_NAME and VALUE columns from the nested table that holds the transformed text.

```
SQL> select b.cust_id, b.cust_gender, b.cust_income_level, b.affinity_card, n.*
             from mining_build_nested_text b,
                  table(b.comments) n
             where b.cust_id = 102998
             order by n.attribute_name;

CUST_ID  C  CUST_INCOME_LEVEL    AFFINITY_CARD  ATTRIBUTE_NAME  VALUE
-------  -- -------------------  -------------  --------------  --------
```

```
102998   M  J: 190,000 - 249,999  1              GOOD             .26894
102998   M  J: 190,000 - 249,999  1              ITEMS             158062
102998   M  J: 190,000 - 249,999  1              KEEP              238765
102998   M  J: 190,000 - 249,999  1              KNOW               .2006
102998   M  J: 190,000 - 249,999  1              LET               299856
102998   M  J: 190,000 - 249,999  1              PURCHASED         142743
102998   M  J: 190,000 - 249,999  1              PURCHASES         173146
102998   M  J: 190,000 - 249,999  1              RECENTLY         .195223
102998   M  J: 190,000 - 249,999  1              SATISFIED        .355851
102998   M  J: 190,000 - 249,999  1              SEVERAL          .355851
102998   M  J: 190,000 - 249,999  1              STORE           .0712537
102998   M  J: 190,000 - 249,999  1              UP               .159838
102998   M  J: 190,000 - 249,999  1              WANTED           .355851
102998   M  J: 190,000 - 249,999  1              WORK             .299856
102998   M  J: 190,000 - 249,999  1              WRITE            .355851
```

The ATTRIBUTE_NAME column holds an item of text from the original comments column. The VALUE column holds the term value. Note that not all words from the original comments column are extracted as terms. For example, the articles the and to are not included.

Steps in the Term Extraction Process

The steps in the term extraction process are summarized in this section. Further details and specific syntax requirements are explained later in this chapter.

Transform a Text Column in the Build Table

First transform the text in the build data. During this process you will generate the text term definitions, which you will reuse for the test and apply data. Perform the following steps:

1. Create an index on the text column in the build table.
2. Create an SVM_CLASSIFIER preference for the index.
3. Define a table to hold the categories specified by the SVM_CLASSIFIER index.
4. Use the FEATURE_PREP table function to create the term definitions and populate an intermediate terms table.
5. Use the FEATURE_EXPLAIN table function to populate the final terms table.
6. Replicate the columns of the original build table (using a view or another table), replacing the text column with a nested table column. Load the terms from the final terms table into the nested table column.

Transform a Text Column in the Test and Apply Tables

The test and apply data must undergo the same pre-processing as the build data. To transform the test and apply data, you will reuse the term definitions generated for the build data. Perform the following steps:

1. Create an index on the text column in the test or apply table.
2. Use the FEATURE_PREP table function to populate an intermediate terms table. Use the term definitions previously generated for the build data.
3. Use the FEATURE_EXPLAIN table function to populate the final terms table.

4. Replicate the columns of the original test or apply table, replacing the text column with a nested table column. Load the terms from the final terms table into the nested table column.

Create the Index and Index Preference

Oracle Text processing requires a text index. Oracle Text supports several types of indexes for querying, cataloging, and classifying text documents. The Oracle Data Mining term extraction process requires a CONTEXT index for text querying.

You must create an index for each text column to be transformed. Use the following syntax to create the index.

```
SQL>CREATE INDEX index_name ON table_name(column_name)
                   INDEXTYPE IS ctxsys.context PARAMETERS ('nopopulate');
```

> **Note:** This statement creates a basic CONTEXT index. You can further define the characteristics of the index by specifying additional arguments to the CREATE INDEX statement. Refer to *Oracle Text Reference* for details.

Oracle Text supports index preferences for overriding the default characteristics of an index. The CREATE_PREFERENCE procedure in the Oracle Text package CTX_DDL creates a preference with the name and type that you specify. The SVM_CLASSIFIER preference type defines the characteristics of an index for Oracle Data Mining.

You must create an index preference when you prepare the build data. It will be reused when you prepare the test and apply data. Use the following syntax to create the index preference.

```
SQL>EXECUTE ctx_ddl.create_preference('preference_name', 'SVM_CLASSIFIER');
```

The SVM_CLASSIFIER index preference uses a predefined table with two numeric columns: ID and CAT. ID holds the case ID; CAT holds the category. The category table is used for internal processing. You must create the category table using the following syntax.

```
SQL>CREATE TABLE category_table_name(id NUMBER, cat NUMBER);
```

Create the Intermediate Terms Table

The FEATURE_PREP table function in the CTXSYS.DRVODM Oracle Text package extracts terms from a text column using an index preference of type SVM_CLASSIFIER. FEATURE_PREP creates a table of term definitions from the build data and reuses these definitions for the test and apply data.

FEATURE_PREP returns an intermediate terms table.

FEATURE_PREP Calling Syntax

FEATURE_PREP is an over-loaded function that accepts two different sets of arguments. You will specify one set of arguments for the build data and another set for the test and apply data.

```
--- syntax for build data ---
            CTXSYS.DRVODM.FEATURE_PREP (
                   text_index                IN   VARCHAR2,
                   case_id                   IN   VARCHAR2,
                   category_tbl              IN   VARCHAR2,
                   category_tbl_id_col       IN   VARCHAR2,
                   category_tbl_cat_col      IN   VARCHAR2,
                   feature_definition_tbl    IN   VARCHAR2,
                   index_preference          IN   VARCHAR2)
                RETURN DRVODM;

--- syntax for test/apply data ---
           CTXSYS.DRVODM.FEATURE_PREP (
                   text_index                IN   VARCHAR2,
                   case_id                   IN   VARCHAR2,
                   feature_definition_tbl    IN   VARCHAR2,
                RETURN DRVODM;
```

FEATURE_PREP Return Value

FEATURE_PREP returns the following columns. The SEQUENCE_ID column holds the case ID; the ATTRIBUTE_ID column holds the term ID.

```
Name                   NULL?   Type
---------------------- ------- ------
SEQUENCE_ID                    NUMBER
ATTRIBUTE_ID                   NUMBER
VALUE                          NUMBER
```

FEATURE_PREP Arguments

FEATURE_PREP accepts the arguments described in Table 4–1.

Table 4–1 FEATURE_PREP Table Function Arguments

Argument Name	Data Type	
text_index	VARCHAR2	Name of the index on the text column in the build, test, or apply table.
case_ID	VARCHAR2	Name of the case ID column in the build, test, or apply table.
category_tbl	VARCHAR2	Name of the table used by the SVM_CLASSIFIER index preference. Specify this argument only for build data.
category_tbl_id_col	VARCHAR2	Specify 'id'. This is the name of the ID column in the table used by the SVM_CLASSIFIER index preference. Specify this argument only for build data.
category_tbl_cat_col	VARCHAR2	Specify 'cat'. This is the name of the CAT column in the table used by the SVM_CLASSIFIER index preference. Specify this argument only for build data.
feature_definition_tbl	VARCHAR2	Name of the term definition table created by FEATURE_PREP. The columns of the term definition table are: Name Null? Type --------------------------------- CAT_ID NUMBER TYPE NUMBER RULE BLOB

Table 4–1 (Cont.) FEATURE_PREP Table Function Arguments

Argument Name	Data Type	
index_preference	VARCHAR2	Name of the SVM_CLASSIFIER index preference. Specify this argument only for build data.

FEATURE_PREP Example

The following example creates an intermediate terms table called txt_term_out. The FEATURE_PREP table function extracts terms from a text column with an index called build_text_idx. The text column is in a build table with a case ID column called cust_id. The index preference txt_pref is applied to the index using the id and cat columns in the table cat_tbl. FEATURE_PREP creates a table of term definitions called txt_pref_terms.

```
CREATE TABLE txt_term_out AS
SELECT *
  FROM TABLE(ctxsys.drvodm.feature_prep (
               'build_text_idx',
               'cust_id',
               'cat_tbl',
               'id',
               'cat',
               'txt_pref_terms',
               'txt_pref'));
```

Create the Final Terms Table

The FEATURE_EXPLAIN table function in the CTXSYS.DRVODM Oracle Text package extracts the term values from the definitions created by FEATURE_PREP and appends the associated word to each value.

FEATURE_EXPLAIN returns the final terms table.

FEATURE_EXPLAIN Calling Syntax

The calling syntax of FEATURE_EXPLAIN is described as follows.

```
            CTXSYS.DRVODM.FEATURE_EXPLAIN (
                   feature_definition_tbl     IN   VARCHAR2,
                RETURN DRVODM;
```

FEATURE_EXPLAIN Return Value

FEATURE_EXPLAIN returns the following columns.

```
Name             Type
---------------   ---------------
text              VARCHAR2(160)
type              NUMBER(3)
ID                NUMBER
score             NUMBER
```

FEATURE_EXPLAIN Arguments

FEATURE_EXPLAIN accepts a single argument: the terms definition table created by FEATURE_PREP.

FEATURE_EXPLAIN Example

The following example creates a final terms table called txt_final_terms using the intermediate terms table txt_term_out. The FEATURE_EXPLAIN table function returns the terms specified in the terms definition table txt_pref_terms.

```
SQL> create table txt_final_terms as
                   select A.sequence_id, B.text, A.value
                         FROM txt_term_out A,
                              TABLE(ctxsys.drvodm.feature_explain(
                                    'txt_pref_terms')) B
                         WHERE A.attribute_id = B.id;
```

Populate a Nested Table Column

Use the final terms table to populate a nested table column of type DM_NESTED_NUMERICALS.

The following example creates the table mining_build_nested_text. (Alternatively, you could create a view.) The table has a case ID column of customer IDs and three customer attribute columns: age, education, and occupation. It also includes a comments column of type DM_NESTED_NUMERICALS created from the terms table txt_final_terms.

```
SQL> CREATE TABLE mining_build_nested_text
       NESTED TABLE comments store AS build_comments
      AS
    SELECT non_text.cust_id,
      non_text.age,
      non_text.education,
      non_text.occupation,
     txt.comments
     FROM
     mining_build_text non_text,
     ( SELECT features.sequence_id,
              cast(COLLECT(dm_nested_numerical(features.text,features.value))
                           as dm_nested_numericals)  comments
       FROM txt_final_terms features
       group by features.sequence_id) txt
     WHERE non_text.cust_id = txt.sequence_id(+);
```

Example: Transforming a Text Column

In the following example, a text column in MINING_BUILD_TEXT is transformed to a nested table column in MINING_BUILD_NESTED_TEXT. The same text column in MINING_APPLY_TEXT is transformed to a nested table column in MINING_APPLY_NESTED_TEXT.

Both MINING_BUILD_TEXT and MINING_APPLY_TEXT have the following columns.

```
 Name                               Null?    Type
 --------------------------------- -------- ---------------------------
 CUST_ID                            NOT NULL NUMBER
 AGE                                         NUMBER
 EDUCATION                                   VARCHAR2(21)
 OCCUPATION                                  VARCHAR2(21)
 COMMENTS                                    VARCHAR2(4000)
```

The following statements create the indexes.

```
SQL> create index build_text_idx on mining_build_text (comments)
             indextype is ctxsys.context parameters ('nopopulate');
```

```
SQL> create index apply_text_idx ON mining_apply_text (comments)
             indextype is ctxsys.context parameters ('nopopulate');
```

The following statements create the index preference and its table.

```
SQL> execute ctx_ddl.create_preference('idx_pref', 'SVM_CLASSIFIER');
SQL> create table idx_pref_cat (id number, cat number);
```

The following statement returns the intermediate terms in the table BUILD_TERMS_OUT. It also creates the table FEATURE_DEFS and populates it with the term definitions.

```
SQL>  create table build_terms_out as
              select * from
                     table (ctxsys.drvodm.feature_prep
                                  ('build_text_idx',
                                   'cust_id',
                                   'idx_pref_cat',
                                   'id',
                                   'cat',
                                   'feature_defs',
                                   'idx_pref'));
```

The following statement returns the final terms in the table BUILD_EXPLAIN_OUT.

```
SQL> create table build_explain_out as
             select a.sequence_id,
                    b.text,
                    a.value
             from build_terms_out a,
             table (ctxsys.drvodm.feature_explain('feature_defs')) b
             where a.attribute_id = b.id;
```

The following statement creates the table MINING_BUILD_NESTED_TEXT. This table contains the non-text attributes from the original build table and a nested table of comments. This table can be used to build a model.

```
SQL> create table mining_build_nested_text
        nested table comments store as build_comments
          as select non_text.cust_id,
                    non_text.age,
                    non_text.education,
                    non_text.occupation,
                    txt.comments
              from mining_build_text non_text,
             (select features.sequence_id,
                cast(collect(dm_nested_numerical(features.text,features.value))
                as dm_nested_numericals)  comments
              from build_explain_out features
              group by features.sequence_id) txt
              where non_text.cust_id = txt.sequence_id(+);
```

The following statement creates the intermediate terms table for the comments column in the apply table, MINING_APPLY_TEXT. It uses the term definitions in the FEATURE_DEFS table, which was created during the pre-processing of the comments column in MINING_BUILD_TEXT.

```
SQL>  create table apply_terms_out as
              select * from
                     table (ctxsys.drvodm.feature_prep
                                  ('apply_text_idx',
                                   'cust_id',
                                   'feature_defs'));
```

The following statement creates the final terms table for apply.

```
SQL> create table apply_explain_out as
             select a.sequence_id,
                    b.text,
                    a.value
             from apply_terms_out a,
             table (ctxsys.drvodm.feature_explain('feature_defs')) b
             where a.attribute_id = b.id;
```

The following statement creates the table MINING_APPLY_NESTED_TEXT. This table contains the non-text attributes from the original apply table and a nested table of comments. This table can be used to apply the model.

```
SQL> create table mining_apply_nested_text
        nested table comments store as apply_comments
          as select non_text.cust_id,
                    non_text.age,
                    non_text.education,
                    non_text.occupation,
                    txt.comments
              from mining_apply_text non_text,
             (select features.sequence_id,
                cast(collect(dm_nested_numerical(features.text,features.value))
                as dm_nested_numericals)  comments
              from apply_explain_out features
              group by features.sequence_id) txt
              where non_text.cust_id = txt.sequence_id(+);
```

5

Building a Model

This chapter explains how to create data mining models and retrieve model details.

Note: This chapter assumes a basic understanding of mining functions and algorithms, as described in *Oracle Data Mining Concepts*.

This chapter contains the following topics:

Steps in Building a Model

The general steps involved in creating a data mining model are summarized as follows:

1. Prepare the data. (See Chapter 3.)
2. Optionally, specify model settings. (See "Model Settings" on page 5-2.)
3. Create the model. (See "Creating a Model" on page 5-6.)
4. View model details. (See "Model Details" on page 5-8.)
5. Test the model. (See *Oracle Data Mining Concepts* for information about test metrics for classification and regression.)
6. Evaluate the model with questions like: How accurate is the model? If there are rules associated with the model, what is their confidence and support? How well does the model address the business question?

Note: To better understand this process, you can look at the source code of the sample data mining programs provided with Oracle Database. See "Sample Mining Models" on page 5-10.

See Also: *Oracle Data Mining Concepts* for more information about the process of building a model

Model Settings

Numerous settings are available for configuring data mining models at build time. To specify settings, create a settings table and pass it to the model creation process. A settings table is optional, because all settings have default values.

Model settings are documented in *Oracle Database PL/SQL Packages and Types Reference*. If you are using the HTML version of *Oracle Data Mining Application Developer's Guide*, the links in Table 5–1 will take you directly to the documentation for each type of model setting.

Table 5–1 Model Settings

Settings	Documentation
Modify the mining function	See *Oracle Database PL/SQL Packages and Types Reference*
Modify the algorithm	See *Oracle Database PL/SQL Packages and Types Reference*
Set global model characteristics	See *Oracle Database PL/SQL Packages and Types Reference*
Enable or disable Automatic Data Preparation	See *Oracle Database PL/SQL Packages and Types Reference*

Specifying a Settings Table

A settings table must have the columns shown in Table 5–2.

Table 5–2 Settings Table Required Columns

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

The values inserted into the `setting_name` column are one or more of several constants defined in the `DBMS_DATA_MINING` package. Depending on what the setting name denotes, the value for the `setting_value` column can be a predefined constant or the actual numerical value corresponding to the setting itself. The `setting_value` column is defined to be `VARCHAR2`. You can explicitly cast numerical inputs to string using the `TO_CHAR()` function, or you can rely on the implicit type conversion provided by the Database.

Example 5–1 creates a settings table for an SVM classification model. Since SVM is not the default classifier, the `ALGO_NAME` setting is used to specify the algorithm. Setting the `SVMS_KERNEL_FUNCTION` to `SVMS_LINEAR` causes the model to be built with a linear kernel. If you do not specify the kernel function, the algorithm chooses the kernel based on the number of attributes in the data.

Example 5–1 Create a Settings Table

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

BEGIN
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svmc_sh_sample_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.svms_kernel_function, dbms_data_mining.svms_linear);
  COMMIT;
END;
```

/

Specifying the Algorithm

The ALGO_NAME setting specifies the algorithm for a model. If you wish to use the default algorithm for a particular mining function, or if there is only one algorithm available for the mining function, you do not need to specify the ALGO_NAME setting.

Table 5–3 Data Mining Algorithms

ALGO_NAME Value	Algorithm	Default?	Mining Model Function
ALGO_AI_MDL	Minimum Description Length	—	attribute importance
ALGO_APRIORI_ASSOCIATION_RULES	Apriori	—	association
ALGO_DECISION_TREE	Decision Tree	—	classification
ALGO_GENERALIZED_LINEAR_MODEL	Generalized Linear Model	—	classification and regression
ALGO_KMEANS	*k*-Means	yes	clustering
ALGO_NAIVE_BAYES	Naive Bayes	yes	classification
ALGO_NONNEGATIVE_MATRIX_FACTOR	Non-Negative Matrix Factorization	—	feature extraction
ALGO_O_CLUSTER	O-Cluster	—	clustering
ALGO_SUPPORT_VECTOR_MACHINES	Support Vector MachineSuppor	yes	default regression algorithm regression, classification, and anomaly detection (classification with no target)

Specifying Costs

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 Table 5–4.

Table 5–4 Cost Matrix Table Required Columns

Column Name	Data Type
actual_target_value	CHAR, VARCHAR2, NUMBER, or FLOAT
predicted_target_value	CHAR, VARCHAR2, NUMBER, or FLOAT
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 5–4, 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.

See Also: *Oracle Data Mining Concepts* for information about costs

Specifying Prior Probabilities

The CLAS_PRIORS_TABLE_NAME setting specifies the name of a table of prior probabilities to be used in building a Naive Bayes model. Prior probabilities can be used to offset differences in distribution between the build data and the actual population. The priors table must have the columns shown in Table 5–5.

Table 5–5 Priors Table Required Columns

Column Name	Data Type
target_value	CHAR, VARCHAR2, NUMBER, or FLOAT
prior_probability	NUMBER

See Also: *Oracle Data Mining Concepts* for information about priors

Specifying Class Weights

The CLAS_WEIGHTS_TABLE_NAME setting specifies the name of a table of class weights to be used to bias a logistic regression (GLM classification) or SVM classification model to favor higher weighted classes. The weights table must have the columns shown in Table 5–6.

Table 5–6 Class Weights Table Required Columns

Column Name	Data Type
target_value	CHAR, VARCHAR2, NUMBER, or FLOAT
class_weight	NUMBER

See Also: *Oracle Data Mining Concepts* for information about class weights

Model Settings in the Data Dictionary

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 Table 5–7.

```
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 5–7 ALL_MINING_MODEL_SETTINGS

Column	Description
owner	Owner of the mining model

Table 5–7 (Cont.) ALL_MINING_MODEL_SETTINGS

Column	Description
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 SVM classification model SVMC_SH_CLAS_SAMPLE. The ALGO_NAME, CLAS_PRIORS_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

7 rows selected.
```

```
SQL> COLUMN setting_value FORMAT A25
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_TYPE
------------------------------ -------------------------     ------------
ALGO_NAME                      ALGO_SUPPORT_VECTOR_MACHINES  INPUT
SVMS_ACTIVE_LEARNING           SVMS_AL_ENABLE                DEFAULT
CLAS_PRIORS_TABLE_NAME         svmc_sh_sample_priors         INPUT
PREP_AUTO                      OFF                           DEFAULT
SVMS_COMPLEXITY_FACTOR         0.244212                      DEFAULT
SVMS_KERNEL_FUNCTION           SVMS_LINEAR                   INPUT
SVMS_CONV_TOLERANCE            .001                          DEFAULT
```

Note: Some model settings are determined by the algorithm if not specified in a settings table. You can find the system-generated setting values by querying the ALL_MINING_MODEL_SETTINGS view.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details about model settings

Creating a Model

The CREATE_MODEL procedure in the DBMS_DATA_MINING package creates a mining model with the specified name, mining function, and case table (build data).

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

See Also: DBMS_DATA_MINING.CREATE_MODEL in *Oracle Database PL/SQL Packages and Types Reference*

Mining Functions

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

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

Supervised functions are predictive. Unsupervised functions are descriptive.

You can specify any of the values in Table 5–8 for the *mining_function* parameter to CREATE_MODEL.

Table 5–8 Mining Model Functions

Mining_Function Value	Description
`ASSOCIATION`	Association is a descriptive mining function. An association model identifies relationships and the probability of their occurrence within a data set. Association models use the Apriori algorithm.
`ATTRIBUTE_IMPORTANCE`	Attribute Importance is a predictive mining function. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome. Attribute Importance models use the Minimal Description Length algorithm.
`CLASSIFICATION`	Classification is a predictive mining function. A classification model uses historical data to predict a categorical target. Classification models can use: Naive Bayes, Decision Tree, Logistic Regression, or Support Vector Machine algorithms. The default is Naive Bayes. The classification function 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 function. A clustering model identifies natural groupings within a data set. Clustering models can use: *k*-Means or O-Cluster algorithms. The default is *k*-Means.
`FEATURE_EXTRACTION`	Feature Extraction is a descriptive mining function. A feature extraction model creates an optimized data set on which to base a model. Feature extraction models use the Non-Negative Matrix Factorization algorithm.
`REGRESSION`	Regression is a predictive mining function. A regression model uses historical data to predict a numerical target. Regression models can use Support Vector Machine or Linear Regression. The default is Support Vector Machine.

See Also: *Oracle Data Mining Concepts* for an introduction to mining functions

Transformation List

You can optionally specify a list of transformations to be applied to the build data before it is acted on by the algorithm. You can use the `STACK` interface in `DBMS_DATA_MINING_TRANSFORM` to build a list of transformation expressions for different attributes, you can specify a single transformation using the `XFORM` interface in `DBMS_DATA_MINING_TRANSFORM`, or you can write your own SQL expressions.

The transformation list argument to `CREATE_MODEL` interacts with the `PREP_AUTO` setting, which controls Automatic Data Preparation (ADP):

- When ADP is on and you specify a transformation list, your transformations are applied with the automatic transformations and embedded in the model.
- 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. This is the pre-release 11 behavior; it is the default behavior in 11*g*.

See Also: *Oracle Data Mining Concepts* for information about Automatic Data Preparation

Model Details

Model details describe model attributes, rules, and other information about the model. You can invoke a `GET_MODEL_DETAILS` function to retrieve model details. A separate `GET_MODEL_DETAILS` function exists for each algorithm.

Model details reverse the transformations applied to the attributes, thus enabling the information to be easily understood by a user. You can obtain the transformations embedded in the model by invoking the `GET_MODEL_TRANSFORMATIONS` function.

Model details, summarized in Table 5–9, support model transparency.

Table 5–9 Model Details

Algorithm	Model Details
Apriori (association rules)	Association rules and frequent itemsets
Decision Tree	The full model with its content and rules
Generalized Linear Models	Attribute-level coefficient and statistics from `GET_MODEL_DETAILS_GLM` and global model information from `GET_MODEL_DETAILS_GLOBAL`
k-Means	For each cluster: statistics and hierarchy information, centroid, attribute histograms, and rules
MDL (attribute importance)	Ranked importance of each attribute
Naive Bayes	Conditional probabilities and priors
Non-Negative Matrix Factorization	Coefficients
O-Cluster	For each cluster: statistics and hierarchy information, centroid, attribute histograms, and rules
Support Vector Machine	Coefficients for linear models

Mining Model Schema Objects

Mining models are database schema objects. Several system and object privileges, described in "Users and Privileges" in *Oracle Data Mining Administrator's Guide*, govern data mining activities. Mining models also support SQL AUDIT and SQL COMMENT, as described in "Mining Model Schema Objects" in *Oracle Data Mining Administrator's Guide*.

Mining Models in the Data Dictionary

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

The columns of ALL_MINING_MODELS are described as follows and explained in Table 5–10.

```
SQL> describe all_mining_models
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                     NOT NULL VARCHAR2(30)
 MODEL_NAME                                NOT NULL VARCHAR2(30)
 MINING_FUNCTION                                    VARCHAR2(30)
 ALGORITHM                                          VARCHAR2(30)
 CREATION_DATE                             NOT NULL DATE
 BUILD_DURATION                                     NUMBER
 MODEL_SIZE                                         NUMBER
 COMMENTS                                           VARCHAR2(4000)
```

Table 5–10 ALL_MINING_MODELS

Column	Description
owner	Owner of the mining model.
model_name	Name of the mining model.
mining_function	The mining model function. See "Mining Functions" on page 5-6.
algorithm	The algorithm used by the mining model. See "Specifying the Algorithm" on page 5-3.
creation_date	The date on which the mining model was created.
build_duration	The duration of the mining model build process in seconds.
model_size	The size of the mining model in megabytes.
comments	Results of a SQL COMMENT applied to the mining model.

The query in Example 5–3 returns information about the mining models in the schema DMUSER.

Example 5–3 ALL_MINING_MODELS

```
SQL> select model_name, mining_function, algorithm, creation_date, build_duration
            FROM all_mining_models WHERE owner IN 'DMUSER';

MODEL_NAME              MINING_FUNCTION         ALGORITHM                  CREATION_DATE BUILD_DURA
---------------------   ---------------------  --------------------------- -----------------------
AI_SH_SAMPLE            ATTRIBUTE_IMPORTANCE   MINIMUM_DESCRIPTION_LENGTH   13-JUN-07         1
AR_SH_SAMPLE            ASSOCIATION_RULES      APRIORI_ASSOCIATION_RULES    13-JUN-07         5
DT_SH_CLAS_SAMPLE       CLASSIFICATION         DECISION_TREE                13-JUN-07         4
KM_SH_CLUS_SAMPLE       CLUSTERING             KMEANS                       13-JUN-07         7
NB_SH_CLAS_SAMPLE       CLASSIFICATION         NAIVE_BAYES                  13-JUN-07         3
OC_SH_CLUS_SAMPLE       CLUSTERING             O_CLUSTER                    13-JUN-07        14
NMF_SH_SAMPLE           FEATURE_EXTRACTION     NONNEGATIVE_MATRIX_FACTOR    13-JUN-07         2
SVMC_SH_CLAS_SAMPLE     CLASSIFICATION         SUPPORT_VECTOR_MACHINES      13-JUN-07         4
GLMR_SH_REGR_SAMPLE     REGRESSION             GENERALIZED_LINEAR_MODEL     13-JUN-07         3
GLMC_SH_CLAS_SAMPLE     CLASSIFICATION         GENERALIZED_LINEAR_MODEL     13-JUN-07         3
SVMR_SH_REGR_SAMPLE     REGRESSION             SUPPORT_VECTOR_MACHINES      13-JUN-07         7
SVMO_SH_CLAS_SAMPLE     CLASSIFICATION         SUPPORT_VECTOR_MACHINES      13-JUN-07         3
T_SVM_CLAS_SAMPLE       CLASSIFICATION         SUPPORT_VECTOR_MACHINES      13-JUN-07         8
T_NMF_SAMPLE            FEATURE_EXTRACTION     NONNEGATIVE_MATRIX_FACTOR    13-JUN-07         7
```

Mining Model Privileges

You need 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.

You can perform specific operations on 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.

> **See Also:** *Oracle Data Mining Administrator's Guide* for details

Sample Mining Models

The models listed in Example 5–3 are created by the Oracle Data Mining sample programs provided with Oracle Database. The sample programs, in PL/SQL and in Java, create mining models that illustrate each of the algorithms supported by Oracle Data Mining.

The sample programs are installed using Oracle Database Companion. Once installed, you can locate them in the rdbms/demo subdirectory under Oracle Home. You can list the sample PL/SQL data mining programs on a Linux system with commands like these.

```
> cd $ORACLE_HOME/rdbms/demo
> ls dm*.sql
```

Likewise, you can list the sample Java data mining programs with commands like the following:

```
> cd $ORACLE_HOME/rdbms/demo
> ls dm*.java
```

> **See Also:** *Oracle Data Mining Administrator's Guide* to learn how to install, configure, and execute the Data Mining sample programs.

6

Scoring and Deployment

This chapter explains how to use data mining models to mine your data.

This chapter contains the following sections:

- In-Database Scoring
- What is Deployment?
- Real-Time Scoring
- Cost-Sensitive Decision Making
- Batch Apply

In-Database Scoring

Scoring, the application of models to new data, is a primary objective of data mining. 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.

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

With Oracle Data Mining, scoring is easy and secure. The scoring engine and the data both reside within the database. Scoring is an extension to the SQL language, so the results of mining can easily be incorporated into applications and reporting systems.

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

What is Deployment?

Deploying a model means using it within a target environment. Model deployment could be:

- Scoring data either for batch or real-time results
- Extracting model details to produce reports. For example: the rules from a Decision Tree model, or the 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 will be used for scoring (export/import)

> **See Also:** *Oracle Data Mining Administrator's Guide* for information about exporting and importing data mining models

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 Example 6–1 uses a Decision Tree model named `dt_sh_clas_sample` to predict the probability that customer 101488 will use an affinity card. A customer representative could retrieve this information in real time when talking to this customer on the phone. Based on the query result, the representative might offer an extra-value card, since there is a 73% chance that the customer will use a card.

Example 6–1 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
--------------
    .727642276
```

Prediction

Oracle Data Mining supports six SQL functions that return results from predictive models (classification or regression).

Predictive models produce a target value for each row (case) in the scoring data. Each SQL function returns different information from the scoring results.

> **See Also:** *Oracle Data Mining Concepts* for information on classification and regression

Best Prediction

(Classification or regression). For classification, the `PREDICTION` function returns the target value that is predicted with the highest probability (or lowest cost, if costs are specified). For regression, `PREDICTION` returns the best predicted target value.

`PREDICTION` supports costs for classification. See "Cost-Sensitive Decision Making" on page 6-5.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses `PREDICTION`

Confidence Bounds (GLM only)

(Classification or regression) The PREDICTION_BOUNDS function returns the upper and lower confidence bounds computed by the model.

Confidence is the degree of certainty that the true value (regression) or probability (classification) lies within the bounded interval. The default confidence is .95. Confidence can be specified by the user in the GLMS_CONF_LEVEL setting for the model. You can override the confidence associated with the model by specifying the confidence inline when you invoke the PREDICTION_BOUNDS function.

No confidence bounds are returned if ridge regression is being used by the algorithm.

See Also:

- *Oracle Database SQL Language Reference* for syntax and an example that uses PREDICTION_BOUNDS
- *Oracle Data Mining Concepts* for information on GLM
- *Oracle Database PL/SQL Packages and Types Reference* for information on GET_MODEL_DETAILS_GLM

Costs

(Classification only) The PREDICTION_COST function returns the cost associated with the class that is predicted with the lowest cost. If you specify a class, the function returns the cost associated with that class.

Costs are a user-specified biasing mechanism for classification. See "Cost-Sensitive Decision Making" on page 6-5.

See Also: *Oracle Database SQL Language Reference* for syntax and an example that uses PREDICTION_COST

Rules (Decision Tree only)

(Classification only) The PREDICTION_DETAILS function returns the rule of a Decision Tree model corresponding to the given prediction. A rule is the condition (combination of attribute values) that leads to a specific classification.

Decision Tree rule identifiers are returned as XML. The full rules can be retrieved with the GET_MODEL_DETAILS_XML function.

See Also:

- *Oracle Database SQL Language Reference* for syntax and an example that uses PREDICTION_DETAILS
- *Oracle Data Mining Concepts* for information about Decision Tree
- *Oracle Database PL/SQL Packages and Types Reference* for information on GET_MODEL_DETAILS_XML

Probability

(Classification only) The PREDICTION_PROBABILITY function returns the probability associated with the best prediction (prediction with the highest probability) or the probability associated with the class that you specify.

See Also: *Oracle Database SQL Language Reference* for syntax and an example that uses PREDICTION_PROBABILITY

Per-Class Results

(Classification only) The PREDICTION_SET function returns all the target classes, associated probabilities, and associated costs (if specified) for each scored row. You can specify parameters to restrict the output of the function.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for syntax and an example that uses PREDICTION_SET
> - "Cost-Sensitive Decision Making" on page 6-5

Clustering

Oracle Data Mining supports three SQL functions that return results when applying clustering models.

Clustering models assign each row to a cluster with an associated probability. Each SQL function returns different information from the scoring results.

> **See Also:** *Oracle Data Mining Concepts* for information on clustering

Cluster Identifier

The CLUSTER_ID function returns the identifier of the cluster predicted with the highest probability.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses CLUSTER_ID

Probability

The CLUSTER_PROBABILITY function returns the probability associated with the cluster to which cases are most likely to be assigned. If you specify a cluster ID, the function returns the probability associated with that cluster.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses CLUSTER_PROBABILITY

Per-Cluster Probabilities

The CLUSTER_SET function returns the probability associated with each cluster for each scored row. You can specify parameters to restrict the output of the function.

Feature Extraction

Oracle Data Mining supports three SQL functions that return results from feature extraction models.

Feature extraction models combine the attributes into a set of features that capture important characteristics of the data. The scoring process generates a value of each feature for each row. The value is a number that identifies the match quality of the case to the feature. Each SQL function returns different information from the scoring results.

Feature Identifier

The FEATURE_ID function returns the identifier of the feature with the highest value (match quality) for the data.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses FEATURE_ID

Match Quality

The FEATURE_VALUE function returns the highest feature value. If you specify a feature ID, the function returns the value of that feature.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses FEATURE_VALUE

Per-Feature Values

The FEATURE_SET function returns the values associated with each feature for each scored row. You can specify parameters to restrict the output of the function.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and an example that uses FEATURE_SET

Save Scoring Results in a Table

If you wish to save the results of a scoring function, you can store them in a table.

This example shows how to save the results of scoring a customer response model.

```
UPDATE CUST_RESPONSE_APPLY_UPDATE
SET prediction = prediction(CUST_RESPONSE19964_DT using *),
     probability =  prediction_probability(CUST_RESPONSE19964_DT using *)
```

The table in question has all of the predictors, and has columns to hold the prediction and probability. The assumption is that any necessary transformations are embedded in the model (otherwise the using clause would need to contain them).

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. The PREDICTION, PREDICTION_COST, and PREDICTION_SET functions all support costs.

A sample cost matrix table is shown in Table 6–1.

Table 6–1 Sample Cost Matrix Table

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	COST
0	0	0
0	1	2
1	0	1
1	1	0

The cost matrix in Table 6–1 specifies costs for a binary target. The matrix indicates that the algorithm should treat a misclassified 0 as twice as costly as a misclassified 1.

If the table name is cost_tbl and it is associated with the Naive Bayes model nb_sh_clas_sample, then the following query takes cost_tbl into account when scoring nb_sh_clas_sample. The output will be 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 (nb_sh_clas_sample COST MODEL
      USING cust_marital_status, education, household_size) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

If there is a possibility that the cost matrix table is not present, or that a cost matrix was not specified for the model, you can use the AUTO keyword with COST MODEL so that scoring only uses costs if the cost matrix is available.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
   FROM mining_data_apply_v
   WHERE PREDICTION (nb_sh_clas_sample COST MODEL AUTO
      USING cust_marital_status, education, household_size) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

You can specify the costs inline when you invoke the scoring function. The inline costs are used for scoring even if a cost matrix table is associated with the model. Here is the same query with the costs specified inline.

```
SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
   FROM mining_data_apply_v
   WHERE PREDICTION (nb_sh_clas_sample
                    COST (0,1) values ((0, 2),
                                       (1, 0))
                    USING cust_marital_status, education, household_size) = 1
   GROUP BY cust_gender
   ORDER BY cust_gender;
```

To associate a cost matrix table with a model for scoring, use the ADD_COST_MATRIX procedure in the DBMS_DATA_MINING package. You can retrieve the cost matrix with the GET_COST_MATRIX function. The REMOVE_COST_MATRIX procedure removes the cost matrix. If you want to use a different cost matrix table with a model, first remove the existing one then add the new one.

```
-- add cost matrix table cost_tbl
-- to model nb_sh_clas_sample
-- for scoring
--
EXEC DBMS_DATA_MINING.ADD_COST_MATRIX('nb_sh_clas_sample', 'cost_tbl');
--
-- replace cost_tbl with cost_tbl_2
--
EXEC DBMS_DATA_MINING.REMOVE_COST_MATRIX('nb_sh_clas_sample', 'cost_tbl');
EXEC DBMS_DATA_MINING.ADD_COST_MATRIX('nb_sh_clas_sample', 'cost_tbl_2');
```

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 scoring the model, the cost matrix used to create the model will be used. If you want to use a different cost matrix table for scoring, first remove the existing cost matrix table then add the new one.

Batch Apply

Oracle Data Mining supports a batch apply operation that writes the results of scoring directly to a table. The columns in the table are mining function-dependent. The apply operation is accomplished by DBMS_DATA_MINING.APPLY.

APPLY creates an output table with the columns shown in Table 6–2.

Table 6–2 APPLY Output Table

Mining Function	Output Columns
classification	CASE_ID PREDICTION PROBABILITY
regression	CASE_ID PREDICTION
anomaly detection (one-class SVM)	CASE_ID PREDICTION PROBABILITY
clustering	CASE_ID CLUSTER_ID PROBABILITY
feature extraction	CASE_ID FEATURE_ID MATCH_QUALITY

Example 6–2 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 6–2 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
```

Comparing APPLY and SQL Scoring Functions

Whether performed by APPLY or by a SQL scoring function, scoring generates the same mining results. 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.

Since APPLY output is stored separately from the scoring data, it must be joined to the scoring data to support queries that include the data attributes being mined (the scored rows). Thus any model that will be used with APPLY must have a case ID.

A case ID is not required for models that will be 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 SQL scoring functions offer flexibility. You can invoke the function that returns the specific information you need. You simply reference a model and identify the kind of mining results you want to retrieve.

7
The Data Mining Java API

This chapter presents an overview of the Oracle Data Mining Java API. The Java API is based on JDM, the industry-standard Java API for data mining.

The Java API is layered on the PL/SQL and SQL language interfaces to Oracle Data Mining. All the SQL-based functionality described in this manual is also implemented in the Java API.

However, the Java API supports several features not implemented in SQL, such as asynchronous execution of mining tasks and text transformation.

> **Note:** The Oracle Data Mining Java API is deprecated in this release.
>
> Oracle recommends that you not use deprecated features in new applications. Support for deprecated features is for backward compatibility only

See Also: *Oracle Data Mining Java API Reference* (javadoc)

This chapter contains the following topics:

- The Java Environment
- Connecting to the Data Mining Engine
- API Design Overview

The Java Environment

The Oracle Data Mining Java API requires Oracle Database 11*g* Release 2 (11.2) and J2SE 1.5. It is backward compatible and can be used with Oracle Database 10.2.

To use the Oracle Data Mining Java API, include the following libraries in your `CLASSPATH`:

```
$ORACLE_HOME/rdbms/jlib/jdm.jar
$ORACLE_HOME/rdbms/jlib/ojdm_api.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/jdbc/lib/ojdbc5.jar
$ORACLE_HOME//oc4j/j2ee/home/lib/connector.jar
$ORACLE_HOME/jlib/orai18n.jar
$ORACLE_HOME/jlib/orai18n-mapping.jar
$ORACLE_HOME/lib/xmlparserv2.jar
```

Connecting to the Data Mining Engine

The Data Mining Engine (DME) is the infrastructure that offers a set of data mining services to its JDM clients. The Oracle Database provides the in-database data mining functionality for JDM through the core Oracle Data Mining option. So in the rest of this document the Oracle Database is referred to as the DME.

To access data mining functionality in the database, a DME Connection needs to be created. To connect to the DME, OJDM supports following different options.

The DME `Connection` object is described in detail in "Features of a DME Connection" on page 7-3.

Connection Factory

The Connection factory is used to create a DME connection. The JDM standard defines ConnectionFactory as a Java interface to provide a vendor neutral approach to create a DME Connection. In this approach, the application infrastructure needs to register the instance of ConnectionFactory in a JNDI server. Applications can lookup for ConnectionFactory in the JNDI server to instantiate a Connection using this factory.

OJDM provides `oracle.dmt.jdm.resource.OraConnectionFactory` class, which can either be instantiated and accessed to create the connection or can be registered in JNDI server. Following code illustrates these two approaches to create a connection factory.

Create ConnectionFactory Using OraConnectionFactory

```
//Create OraConnectionFactory
javax.datamining.resource.ConnectionFactory connFactory =
                 oracle.dmt.jdm.resource.OraConnectionFactory();
```

Lookup ConnectionFactory From the JNDI Server

```
//Setup the initial context to connect to the JNDI server
Hashtable env = new Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY,
"oracle.dmt.jdm.resource.OraConnectionFactory" );
env.put( Context.PROVIDER_URL, "http://myHost:myPort/myService" );
env.put( Context.SECURITY_PRINCIPAL, "user" );
env.put( Context.SECURITY_CREDENTIALS, "password" );
InitialContext jndiContext = new javax.naming.InitialContext( env );
// Perform JNDI lookup to obtain the connection factory
javax.datamining.resource.ConnectionFactory dmeConnFactory =
(ConnectionFactory) jndiContext.lookup("java:comp/env/jdm/MyServer");
//Lookup ConnectionFactory
javax.datamining.resource.ConnectionFactory connFactory =
  (ConnectionFactory) jndiContext.lookup("java:comp/env/jdm/MyServer");
```

Connect Using JDBC

This option is useful when the applications want to control the JDBC Connections outside the OJDM and allow the OraConnectionFactory to use the specified OracleDataSource to create the database connection. This approach gives applications the ability to use the implicit connection caching features as required. By default, OJDM doesn't enable the implicit connection caching. *Oracle Database JDBC Developer's Guide* for information about connection caching.

```
//Create an OracleDataSource
OracleDataSource ods = new OracleDataSource();
```

```
ods.setURL(URL);
ods.setUser(user);
ods.setPassword(password);

//Create a connection factory using the OracleDataSource
javax.datamining.resource.ConnectionFactory connFactory =
  oracle.dmt.jdm.resource.OraConnectionFactory(ods);
//Create DME Connection
javax.datamining.resource.Connection dmeConn =
    connFactory.getConnection();
```

Connect Using ConnectionSpec

This option is useful when the application doesn't want to pre-create the JDBC Connection and allow OJDM to maintain the JDBC Connection. Here the user needs to create an empty ConnectionSpec instance using getConnectionSpec() method in the oracle.dmt.jdm.resource.OraConnectionFactory class and create a DME Connection using the connection spec. The following code illustrates the usage.

```
//Create ConnectionSpec
ConnectionSpec connSpec = m_dmeConnFactory.getConnectionSpec();
connSpec.setURI("jdbc:oracle:thin:@host:port:sid");
connSpec.setName("user");
connSpec.setPassword("password");

//Create DME Connection
javax.datamining.resource.Connection m_dmeConn =
m_dmeConnFactory.getConnection(connSpec);
```

Features of a DME Connection

In the Oracle Data Mining Java API, the DME Connection is the primary factory object. The Connection instantiates the object factories using the getFactory method. The Connection object provides named object lookup, persistence, and task execution features.

Create Object Factories

The Connection.getFactory method creates a factory object. For example, to create a factory for the PhysicalDataSet object, pass the absolute name of the object to this method. The getFactory method creates an instance of PhysicalDataSetFactory.

```
javax.datamining.data.PhysicalDataSetFactory pdsFactory =
                dmeConn.getFactory("javax.datamining.data.PhysicalDataSet");
```

Provide Access to Mining Object Metadata

The Connection object provides methods for retrieving metadata about mining objects.

Method	Description
getCreationDate	Returns the creation date of the specified named object. `getCreationDate(java.lang.String objectName, NamedObject objectType) returns java.util.Date`

Method	Description
`getDescription`	Returns the description of the specified mining object. `getDescription(java.lang.String objectName, NamedObject objectType) returns java.lang.String`
`getObjectNames`	Returns a collection of the names of the objects of the specified type. `getObjectNames(NamedObject objectType) returns java.util.Collection`

`getObjectNames`	This is an Oracle JDM extension method that is added in 11.1 to provide a listing of mining object names across schemas or within a schema. It provides various optional method arguments that can be used to get a filtered list of arguments.
	`getObjectNames(`
	`java.lang.String schemaPattern,`
	`NamedObject objectType,`
	`java.lang.String objectNamePattern,`
	`javax.datamining.Enum minorType_1,`
	`javax.datamining.Enum minorType_2 ):`
	`returns java.sql.ResultSet`
	See Example 7–1.

Example 7–1 Oracle JDM Extension Method getObjectNames

This example illustrates the `getObjectNames` method.

To list the names of classification test metrics computed by the user `SCOTT`, specify:

the `schemaPattern` as "SCOTT"
`objectType` as `NamedObject.testMetrics`
`objectPattern` as null
`minorType_1` as `MiningFunction.classification`
`minorType_2` as null

Irrespective of the type of filters specified, the `getObjectNames` method returns the `java.sql.ResultSet` object with the following columns.

Column Name	Data Type	Description
`SCHEMA_NAME`	String	Name of the schema (can be null)
`TYPE`	String	Type of the mining object
`NAME`	String	Name of the mining object
`MINOR_TYPE_1`	String	Mining objects can have minor/sub types. For example, model objects can have function and algorithm as minor types.
`MINOR_TYPE_2`	String	Mining objects can have more than one minor type. If they have a second minor type, then this column is used.
`CREATION_DATE`	Timestamp	Date when this object was created
`DESCRIPTION`	String	Description of the object

Persistence and Retrieval of Mining Objects

The `Connection` object provides methods for retrieving mining objects and saving them in the DME. Persistent objects are stored as database objects. Transient objects are stored in memory by the `Connection` object.

Method	Description
`saveObject`	Saves the named object in the metadata repository associated with the connection. `saveObject(java.lang.String name, MiningObject object, boolean replace)`
`retrieveObject`	Retrieves a copy of the specified named object from the metadata repository associated with the connection. `retrieveObject(java.lang.String objectIdentifier) returns MiningObject`
`retrieveObject`	Retrieves a copy of the named object from the metadata repository associated with the connection. `retrieveObject(java.lang.String name, NamedObject objectType) returns MiningObject`
`retrieveObjects`	Returns a collection of mining objects of the given type that were created within the specified time interval (from createAfter to createBefore). `(java.util.Date createdAfter, java.util.Date createdBefore, NamedObject objectType): returns java.util.Collection`
`retrieveObjects`	Returns a collection of mining objects of the specified type that were created within the specified time interval (from createAfter to createBefore) `retrieveObjects(java.util.Date createdAfter, java.util.Date createdBefore, NamedObject objectType, Enum minorType): returns java.util.Collection`

See Also:

- Chapter 2, "A Tour of the Data Mining APIs".
- "API Design Overview" on page 7-7.

Execute Mining Tasks

The `Connection` object provides an `execute` method, which can execute mining tasks either asynchronously or synchronously. The DME uses the database Scheduler to execute mining tasks, which are stored in the user's schema as Scheduler jobs. The following methods are used to execute the tasks.

Task Execution	execute method syntax
asynchronous	`execute(java.lang.String taskName) returns ExecutionHandle`
synchronous	`execute(Task task,java.lang.Long timeout)) returns ExecutionHandle` Typically to be used with single record scoring, but it may be used in other contexts as well.

See Also:

- "Task" on page 2-7
- "Executing Mining Tasks" on page 7-10
- *Oracle Database Administrator's Guide* for information about the Oracle Database Scheduler.

Retrieve DME Capabilities and Metadata

The `Connection` interface provides a `ConnectionMetaData` and `supportsCapability` retrieval methods. This feature is useful for applications to know more about the DME at runtime. The following methods are used for retrieving this information from the connection.

Method	Description

`getMetaData`	Returns information about the underlying DME instance represented through an active connection. `ConnectionMetaData` provides version information for the JDM implementation and Oracle Database.
	`getMetaData()`
	`returns ConnectionMetaData`

`getSupportedFunctions`	Returns an array of mining functions that are supported by the implementation.
	`getSupportedFunctions()`
	`returns MiningFunction[]`

`getSupportedAlgorithms`	Returns an array of mining algorithms that are supported by the specified mining function.
	`getSupportedAlgorithms(MiningFunction function)`
	`returns MiningAlgorithm[]`

`supportsCapability`	Returns true if the specified combination of mining capabilities is supported. If an algorithm is not specified, returns true if the specified function is supported.
	`supportsCapability(MiningFunction function,`
	`MiningAlgorithm algorithm,`
	`MiningTask taskType)`
	`returns boolean`

Retrieve Version Information

The `ConnectionMetaData` object provides methods for retrieving JDM standard version information and Oracle version information.

Method	Description

`getVersion`	Returns the version of the JDM Standard API. It must be "JDM 1.0" for the first release of JDM.
	`getVersion()`
	`returns String`

`getMajorVersion`	Returns the major version number. For the first release of JDM, this is "1".
	`getMajorVersion()`
	`returns int`

Method	Description

`getMinorVersion`	Returns the minor version number. For the first release of JDM, this is "0".
	`getMinorVersion()`
	`     returns int`

`getProviderName`	Returns the provider name as "Oracle Corporation".
	`getProviderName()`
	`     returns String`

`getProviderVersion`	Returns the version of the Oracle Database that shipped the Oracle Data Mining Java API jar file.
	`getProviderVersion()`
	`     returns String`

`getProviderDMEVersion`	Returns the DME version of the provider
	`getProviderDMEVersion()`
	`     returns String`

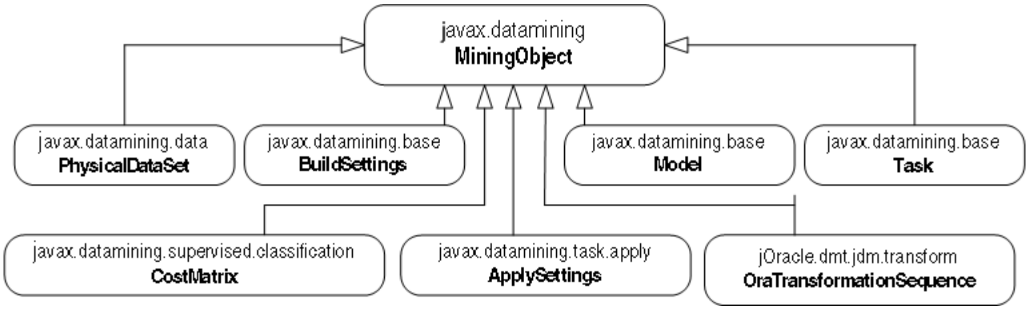
API Design Overview

This section briefly describes the OJDM design principles to familiarize the developers with the API. The JDM standard uses the factory method pattern as the core design pattern for the API. User can instantiate a JDM object using its factory. This enables JDM vendors like Oracle to implement a vendor neutral API. OJDM follows the same factory method pattern for its extensions. `javax.datamining` is the base package for the JDM standard defined classes and **oracle.dmt.jdm** is the base package for the Oracle extensions to the JDM standard

The JDM standard organizes its packages by the mining functions and mining algorithms. For example, **javax.datamining.supervised** package has all the supervised functions related classes and sub-packages **java.datamining.supervised.classification** and **java.datamining.supervised.regression**. Each function sub-package has the classes related to that function. Similarly, **javax.datamining.algorithm** is the base package for all algorithms and each algorithm has its sub-package under this package, for example, **javax.datamining.algorithm.naivebayes** is the sub-package for Naïve Bayes algorithm related classes. OJDM follows a similar package structure for the extensions, for example, feature extraction is a non-JDM standard function supported by the OJDM, here **oracle.dmt.jdm.featureextraction** is the package for this function and **oracle.dmt.jdm.algorithm.nmf** package for the Non-Negative Matrix Factorization algorithm used for feature extraction.

The JDM standard has some core packages that define common classes and packages for tasks, model details, rules and statistics. For more details refer to the JDM javadoc. The class diagram in Figure 7–1 illustrates the inheritance hierarchy of the named mining objects that are discussed in Chapter 2. In the subsequent sections more class diagrams are used to illustrate other OJDM objects. Note that the classes/interfaces shown in gray color are oracle JDM extension interfaces/classes. In Figure 7–1, **oracle.dmt.jdm.transform.OraTransformationSequence** is an Oracle extension to the mining objects defined in JDM 1.1 standard.

Figure 7–1 JDM Named Objects Class Diagram

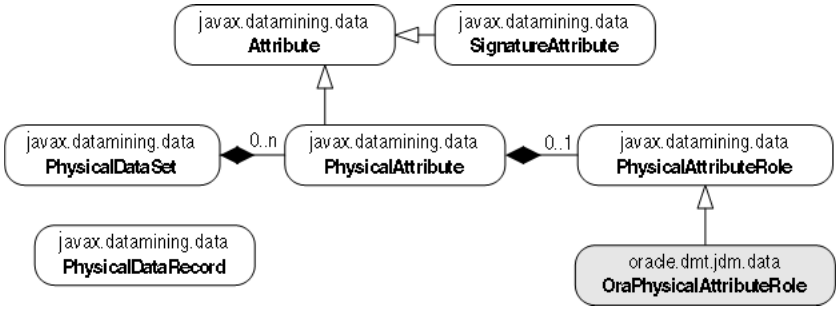


Describing the Mining Data

The JDM standard defines physical and logical data objects to describe the mining attribute characteristics of the data as well as statistical computations for describing the data.

The `javax.datamining.data` package contains all the data-related classes. The class diagram in Figure 7–2 illustrates the class relationships of the data objects supported by the Oracle Data Mining Java API.

Figure 7–2 Data Objects in Oracle Data Mining Java API



The `PhysicalDataSet` object is used to specify the name and location of the dataset used for mining operations. For example, to represent a model build input dataset `MINING_DATA_BUILD_V` in a `DMUSER` schema account, `PhysicalDataSet` object is created with the data `URI DMUSER.MINING_DATA_BUILD_V`. The schema name prefix is optional when accessing the datasets in the same user account.

Note that in the class diagram in Figure 7–2 a `PhysicalDataSet` can have `PhysicalAttribute` objects. A `PhysicalAttribute` represents physical characteristics of the columns of the input dataset; optionally physical attributes can specify the roles of the column. For example, in the `MINING_DATA_BUILD_V` dataset, `CUST_ID` uniquely identifies each case used for mining. So the role of the `CUST_ID` column is specified as case id.

Example 7–2 illustrates the code sample that creates the `PhysicalDataSet` object. Note that the `PhysicalDataSet` object is saved with the name `JDM_BUILD_PDS` that can be specified as input to model build that we discuss later in "Build Settings" on page 7-9

The `PhysicalDataRecord` object shown in Figure 7–2 is used to specify a single record of a dataset. It is used for single record apply that we will discuss in a later section of this Chapter.

The SignatureAttribute is used to specify the model signature of a mining model that will be discussed in later section of this Chapter.

In OJDM attribute data types are used to implicitly specify the mining attribute types. For example, all VARCHAR2 columns are treated as categorical and all NUMBER columns are treated as numerical. So there is no need to specify logical data details in OJDM. However, to rename attributes of a column, a user can specify the embedded transformations that are discussed in the next section.

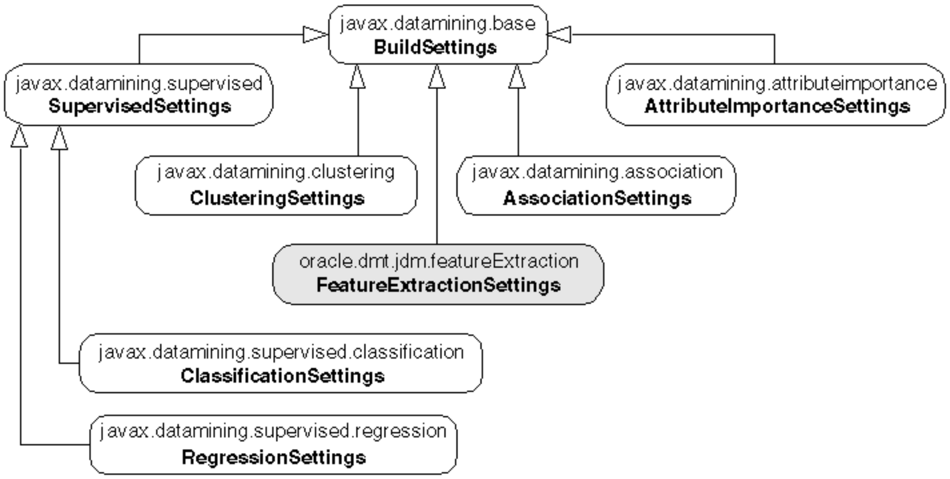
Example 7–2 Creation of a Physical Dataset

```
//Create PhysicalDataSetFactory
PhysicalDataSetFactory pdsFactory =
(PhysicalDataSetFactory)m_
dmeConn.getFactory("javax.datamining.data.PhysicalDataSet");
//Create a PhysicalDataSet object
PhysicalDataSet buildData = pdsFactory.create("DMUSER.MINING_DATA_BUILD_V",
false);
//Create PhysicalAttributeFactory
PhysicalAttributeFactory paFactory =
(PhysicalAttributeFactory)m_
dmeConn.getFactory("javax.datamining.data.PhysicalAttribute");
//Create PhysicalAttribute object
PhysicalAttribute pAttr = paFactory.create(
"cust_id", AttributeDataType.integerType, PhysicalAttributeRole.caseId );
//Add the attribute to the PhysicalDataSet object
buildData.addAtribute(pAttr);
//Save the physical data set object
dmeConn.saveObject("JDM_BUILD_PDS", buildData, true);
```

Build Settings

In the Oracle Data Mining Java API, the BuildSettings object is saved as a table in the database. The settings table is compatible with the DBMS_DATA_MINING.CREATE_MODEL procedure. The name of the settings table must be unique in the user's schema. Figure 7–3 illustrates the build settings class hierarchy.

Figure 7–3 Build Settings Class Diagram.


javax.datamining.base
BuildSettings
javax.datamining.supervised
SupervisedSettings
javax.datamining.attributeimportance
AttributeImportanceSettings
javax.datamining.clustering
ClusteringSettings
javax.datamining.association
AssociationSettings
oracle.dmt.jdm.featureExtraction
FeatureExtractionSettings
javax.datamining.supervised.classification
ClassificationSettings
javax.datamining.supervised.regression
RegressionSettings

The code in Example 7–3 illustrates the creation and storing of a classification settings object with a tree algorithm.

Example 7–3 Creation of a Classification Settings with Decision Tree Algorithm

```
//Create a classification settings factory
ClassificationSettingsFactory clasFactory =
(ClassificationSettingsFactory)dmeConn.getFactory
     ("javax.datamining.supervised.classification.ClassificationSettings");
//Create a ClassificationSettings object
ClassificationSettings clas = clasFactory.create();
//Set target attribute name
clas.setTargetAttributeName("AFFINITY_CARD");
//Create a TreeSettingsFactory
TreeSettingsFactory treeFactory =
(TreeSettingsFactory)dmeConn.getFactory
     ("javax.datamining.algorithm.tree.TreeSettings");
//Create TreeSettings instance
TreeSettings treeAlgo = treeFactory.create();
treeAlgo.setBuildHomogeneityMetric(TreeHomogeneityMetric.entropy);
treeAlgo.setMaxDepth(10);
treeAlgo.setMinNodeSize( 10, SizeUnit.count );
//Set algorithm settings in the classification settings
clas.setAlgorithmSettings(treeAlgo);
//Save the build settings object in the database
dmeConn.saveObject("JDM_TREE_CLAS", clas, true);
```

Enable Automated Data Preparation

In 11.1, all mining algorithms support automated data preparations (ADP). By default for decision tree and GLM algorithms, ADP is enabled. For other algorithms it is disabled by default for backward compatibility reasons. To enable ADP explicitly for the algorithms that do not enable by default, invoke the following function, by specifying the `useAutomatedDataPreparations` boolean flag as true.

```
OraBuildSettings.useAutomatedDataPreparations
          (boolean useAutomatedDataPreparations)
```

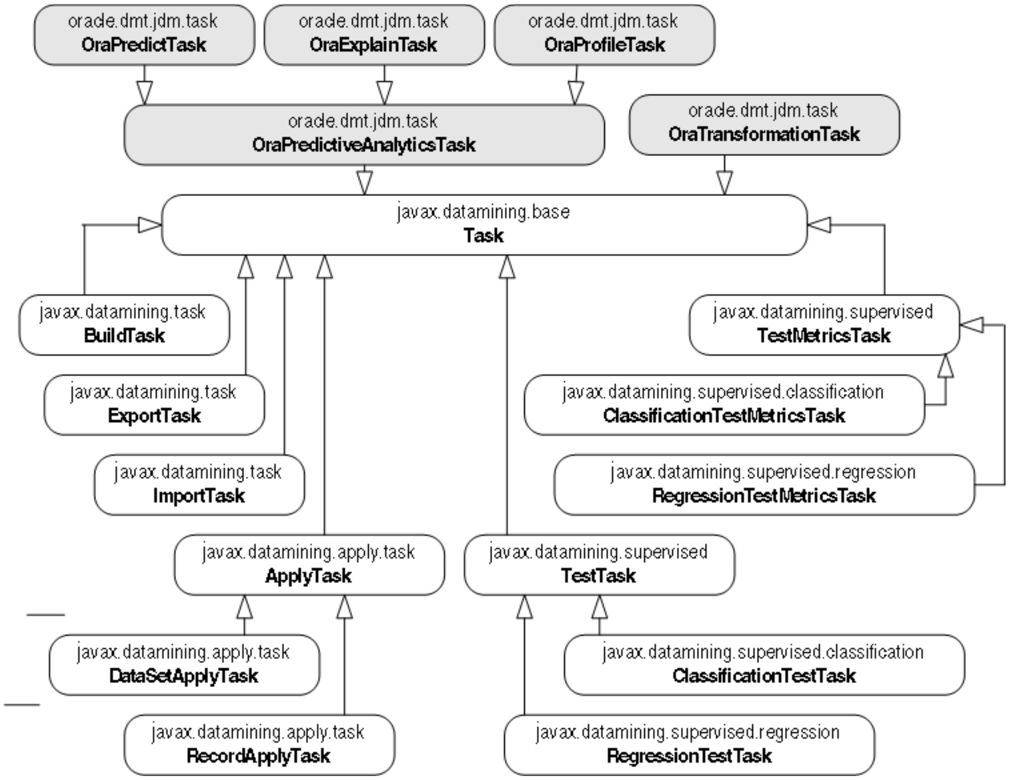
For more information about automatic data preparation, see *Oracle Data Mining Concepts*.

Executing Mining Tasks

OJDM uses the `DBMS_SCHEDULER` infrastructure for executing mining tasks either synchronously or asynchronously in the database. A mining task is saved as a `DBMS_SCHEDULER` job in the user schema and is set to `DISABLED` state. When user calls the execute method in DME Connection, the job state will be changed to `ENABLED` and scheduler starts executing the mining task by creating a new database session for asynchronous executions. For synchronous executions scheduler uses the same database session opened by the DME connection.

The class diagram in Figure 7–4 illustrates the different types of tasks that are available in OJDM and its class hierarchy. Subsequent sections will discuss more about the individual tasks shown in this diagram

Figure 7–4 Task Class Diagram



DBMS_SCHEDULER provides additional scheduling and resource management features. You can extend the capabilities of Oracle Data Mining tasks by using the Scheduler infrastructure.

> **See Also:** *Oracle Database Administrator's Guide* for information about the database scheduler.

Creating Mining Task Workflows

In Oracle Data Mining 11.1, the task infrastructure supports applications to specify dependent tasks through the API and deploy the execution of the tasks to the database server. The server executes complete workflow of tasks specified through the API and once deployed it does not depend on client. Client can monitor the execution process using OJDM API. For example, typically after the completion of data preparations, model is built and then tested and applied. Both test and apply can be done in parallel after model build is successful.

To build a task flow invoke the method OraTask.addDependency(String parentTaskName). For example, the code in Example 7–4 illustrates how to setup a mining task workflow, where first run the transformations task and then model build task. After successful completion of the build task run apply and test tasks in parallel.

Example 7–4 Building Mining Task Workflows

```
//Task objects declarations
private TransformationTask xformTask;
private BuildTask buildTask;
private TestTask testTask;
private DataSetApplyTask applyTask;
```

```
//Creation of the tasks and task input objects are skipped for this example
…
//Save the first task in the workflow (the transformations task)
dmeConn.saveObject("transformationTask", xformTask, true);
//Specify dependencies before saving of the tasks
buildTask.addDependency("transformationTask");
dmeConn.saveObject("modelBuildTask", buildTask, true);
testTask.addDependency("modelBuildTask");
dmeConn.saveObject("modelTestTask", testTask, true);
applyTask.addDependency("modelBuildTask");
dmeConn.saveObject("modelApplyTask", applyTask, true);
//Execute the first task in the workflow to initiate the execution of the whole
workflow
dmeConn.execute("transformationTask");
```

Building a Mining Model

The `javax.datamining.task.BuildTask` class is used to build a mining model. Prior to building a model, a `PhysicalDataSet` object and a `BuildSettings` object must be saved.

Example 7–5 illustrates the building of a tree model using the `PhysicalDataSet` described in "Describing the Mining Data" on page 7-8 and the `BuildSettings` described in "Build Settings" on page 7-9.

Example 7–5 Building a Model

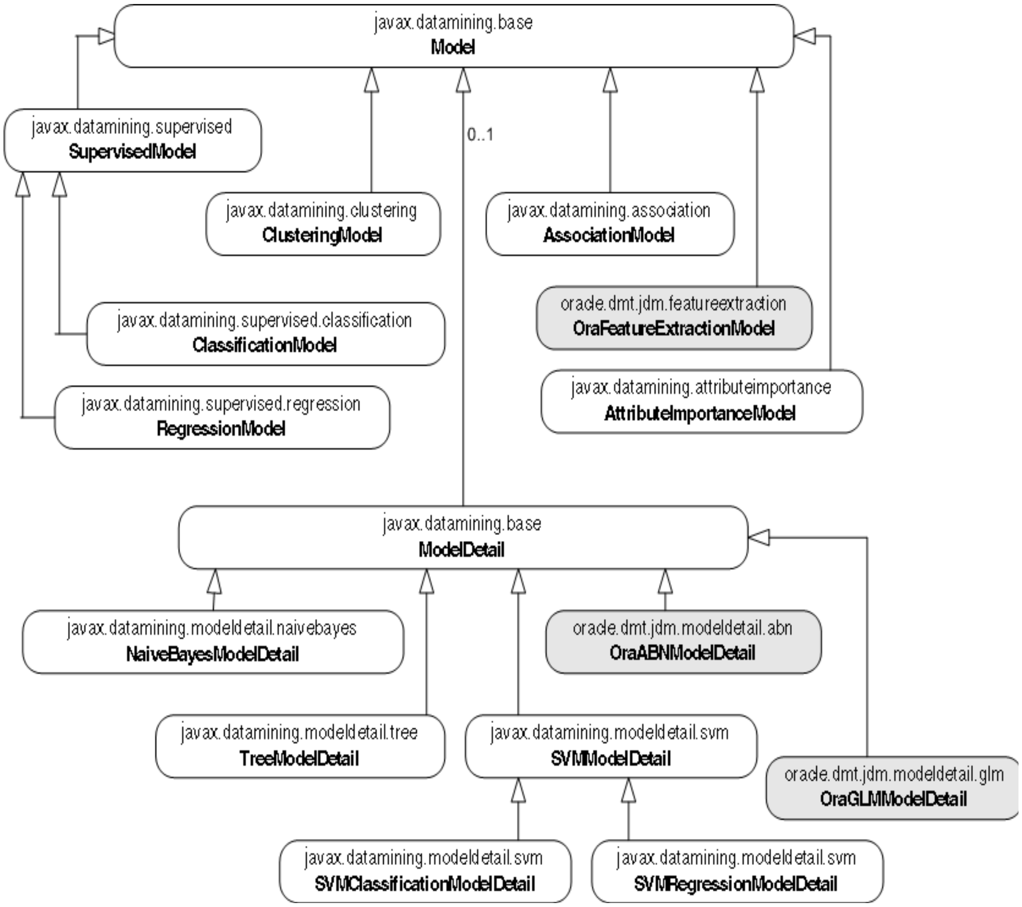
```
//Create BuildTaskFactory
BuildTaskFactory buildTaskFactory =
     dmeConn.getFactory("javax.datamining.task.BuildTask");
//Create BuildTask object
BuildTask buildTask = buildTaskFactory.create
     ( "JDM_BUILD_PDS","JDM_TREE_CLAS","JDM_TREE_MODEL");
//Save BuildTask object
dmeConn.saveObject("JDM_BUILD_TASK", buildTask, true);
//Execute build task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_BUILD_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
```

Exploring Model Details

After building a model using the `BuildTask`, a model object is persisted in the database. It can be retrieved to explore the model details.

The class diagram in Figure 7–5 illustrates the different types of model objects and model details objects supported by the Oracle Data Mining Java API.

Figure 7–5 Model and Model Detail Class Diagram


javax.datamining.base
Model
0..1
javax.datamining.supervised
SupervisedModel
javax.datamining.clustering
ClusteringModel
javax.datamining.association
AssociationModel
oracle.dmt.jdm.featureextraction
OraFeatureExtractionModel
javax.datamining.supervised.classification
ClassificationModel
javax.datamining.attributeimportance
AttributeImportanceModel
javax.datamining.supervised.regression
RegressionModel
javax.datamining.base
ModelDetail
javax.datamining.modeldetail.naivebayes
NaiveBayesModelDetail
oracle.dmt.jdm.modeldetail.abn
OraABNModelDetail
javax.datamining.modeldetail.tree
TreeModelDetail
javax.datamining.modeldetail.svm
SVMModelDetail
oracle.dmt.jdm.modeldetail.glm
OraGLMModelDetail
javax.datamining.modeldetail.svm
SVMClassificationModelDetail
javax.datamining.modeldetail.svm
SVMRegressionModelDetail

Example 7–6 illustrates the retrieval of the classification tree model built in "Building a Mining Model" on page 7-12 and its `TreeModelDetail`.

Example 7–6 Retrieve Model Details

```
//Retrieve classification model from the DME
ClassificationModel treeModel = (ClassificationModel)dmeConn.retrieveObject
     ( "JDM_TREE_MODEL", NamedObject.model);
//Retrieve tree model detail from the model
TreeModelDetail treeDetail = (TreeModelDetail)treeModel.getModelDetail();
//Get the root node
TreeNode rootNode = treeDetail.getRootNode();
//Get child nodes
TreeNode[] childNodes = rootNode.getChildren();
//Get details of the first child node
int nodeId = childNodes[0].getIdentifier();
long caseCount = childNodes[0].getCaseCount();
Object prediction = childNodes[0].getPrediction();
```

Testing a Model

Once a supervised model has been built, it can be evaluated using a test operation. The JDM standard defines two types of test operations: one that takes the mining model as input, and the other that takes the apply output table with the actual and predicted value columns.

`javax.datamining.supervised.TestTask` is the base class for the model- based test tasks, and `javax.datamining.supervised.TestMetricsTask` is the base class for the apply output table-based test tasks.

The test operation creates and persists a test metrics object in the DME. For classification model testing, either of the following can be used.

```
javax.datamining.supervised.classification.ClassificationTestTask
javax.datamining.supervised.classification.ClassificationTestMetricsTask
```

Both of these tasks create the named object `javax.datamining.supervised.classification.ClassificationTestMetrics` and store it as a table in the user's schema.

The classification test metrics components, confusion matrix, lift results, and ROC associated with the `ClassificationTestMetrics` object are stored in separate tables whose names are the `ClassificationTestMetrics` object name followed by the suffix `_CFM`, `_LFT`, or `_ROC`. These tables can be used to display test results in dashboards, BI platforms such as Oracle BI, Business Objects, and so on.

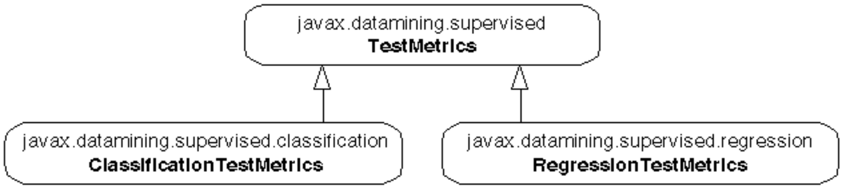
Similarly for regression model testing, either of the following can be used:

```
javax.datamining.supervised.regression.RegressionTestTask
javax.datamining.supervised.regression.RegressionTestMtericsTask
```

Both these tasks create a named object `javax.datamining.supervised.regression.RegressionTestMetrics` and store it as a table in the user's schema.

The class diagram in Figure 7–6 illustrates the test metrics class hierarchy. It refers to Figure 7–4, "Task Class Diagram" on page 7-11 for the class hierarchy of test tasks.

Figure 7–6 Test Metrics Class Hierarchy


javax.datamining.supervised
TestMetrics
javax.datamining.supervised.classification
ClassificationTestMetrics
javax.datamining.supervised.regression
RegressionTestMetrics

Example 7–7 illustrates the test of a tree model `JDM_TREE_MODEL` using the `ClassificationTestTask` on the dataset `MINING_DATA_TEST_V`.

Example 7–7 Testing a Model

```
//Create & save PhysicalDataSpecification
PhysicalDataSet testData = m_pdsFactory.create(
        "MINING_DATA_TEST_V", false );
PhysicalAttribute pa = m_paFactory.create("cust_id",
        AttributeDataType.integerType, PhysicalAttributeRole.caseId );
testData.addAttribute( pa );
m_dmeConn.saveObject( "JDM_TEST_PDS", testData, true );
//Create ClassificationTestTaskFactory
ClassificationTestTaskFactory testTaskFactory =
  (ClassificationTestTaskFactory)dmeConn.getFactory(
     "javax.datamining.supervised.classification.ClassificationTestTask");
//Create, store & execute Test Task
ClassificationTestTask testTask = testTaskFactory.create(
        "JDM_TEST_PDS", "JDM_TREE_MODEL", "JDM_TREE_TESTMETRICS" );
testTask.setNumberOfLiftQuantiles(10);
testTask.setPositiveTargetValue(new Integer(1));
//Save TestTask object
dmeConn.saveObject("JDM_TEST_TASK", testTask, true);
//Execute test task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_TEST_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
//Explore the test metrics after successful completion of the task
if(ExecutionState.success.equals(execStatus.getState())) {
  //Retrieve the test metrics object
  ClassificationTestMetrics testMetrics =
          (ClassificationTestMetrics)dmeConn.getObject("JDM_TREE_TESTMETRICS");
  //Retrieve confusion matrix and accuracy
  Double accuracy = testMetrics.getAccuracy();
  ConfusionMatrix cfm = testMetrics.getConfusionMatrix();
  //Retrieve lift
  Lift lift = testMetrics.getLift();
  //Retrieve ROC
  ReceiverOperatingCharacterics roc = testMetrics.getROC();
}
```

In Example 7–7, a test metrics object is stored as a table called JDM_TREE_TESTMETRICS. The confusion matrix is stored in the JDM_TREE_TESTMETRICS_CFM table, lift is stored in the JDB_TREE_TESTMETRICS_LFT table, and ROC is stored in the JDM_TREE_TESTMETRICS_ROC table. You can use BI tools like Oracle Discoverer to query these tables and create reports.

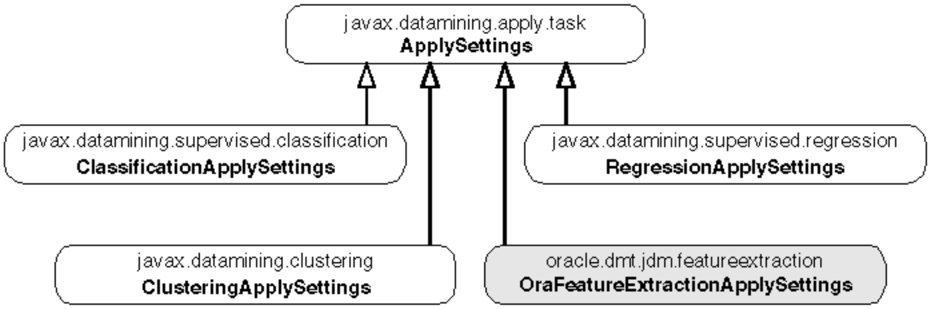
Applying a Model for Scoring Data

All supervised models can be applied to data to find the prediction. Some of the unsupervised models, such as clustering and feature extraction, support the apply operation to find the cluster id or feature id for new records.

The JDM standard API provides an ApplySettings object to specify the type of output for the scored results. javax.datamining.task.apply.ApplySettings is the base class for all apply settings. In the Oracle Data Mining Java API, the ApplySettings object is transient; it is stored in the Connection context, not in the database.

The class diagram in Figure 7–7 illustrates the class hierarchy of the apply settings available in the Oracle Data Mining Java API.

Figure 7–7 Apply Settings


javax.datamining.apply.task
ApplySettings
javax.datamining.supervised.classification
ClassificationApplySettings
javax.datamining.supervised.regression
RegressionApplySettings
javax.datamining.clustering
ClusteringApplySettings
oracle.dmt.jdm.featureextraction
OraFeatureExtractionApplySettings

In the Oracle Data Mining Java API, default apply settings produce the apply output table in fixed format. The list in Table 7–1 illustrates the default output formats for different functions.

Table 7–1 Default Output Formats for Different Functions

Mining Function				
Classification without Cost	Case ID	Prediction	Probability	
Classification with Cost	Case ID	Prediction	Probability	Cost
Regression	Case ID	Prediction		
Feature extraction	Case ID	Feature ID	Value	

All types of apply settings support source and destination attribute mappings. For example, if the original apply table has customer name and age columns that need to be carried forward to the apply output table, it can be done by specifying the source destination mappings in the apply settings.

In the Oracle Data Mining Java API, classification apply settings support map by rank, top prediction, map by category, and map all predictions. Regression apply settings support map prediction value. Clustering apply settings support map by rank, map by cluster id, map top cluster, and map all clusters. Feature extraction apply settings support map by rank, map by feature id, map top feature, and map all features.

Example 7–8 illustrates the applying of a tree model `JDM_TREE_MODEL` using `ClassificationApplyTask` on the dataset `MINING_DATA_APPLY_V`.

Example 7–8 Applying a Model

```
//Create & save PhysicalDataSpecification
PhysicalDataSet applyData = m_pdsFactory.create( "MINING_DATA_APPLY_V", false );
PhysicalAttribute pa = m_paFactory.create("cust_id",
        AttributeDataType.integerType, PhysicalAttributeRole.caseId );
applyData.addAttribute( pa );
m_dmeConn.saveObject( "JDM_APPLY_PDS", applyData, true );
//Create ClassificationApplySettingsFactory
ClassificationApplySettingsFactory applySettingsFactory =
  (ClassificationApplySettingsFactory)dmeConn.getFactory(
     "javax.datamining.supervised.classification. ClassificationApplySettings");
//Create & save ClassificationApplySettings
ClassificationApplySettings clasAS = applySettingsFactory.create();
m_dmeConn.saveObject( "JDM_APPLY_SETTINGS", clasAS, true);
//Create DataSetApplyTaskFactory
DataSetApplyTaskFactory applyTaskFactory =
  (DataSetApplyTaskFactory)dmeConn.getFactory(
     "javax.datamining.task.apply.DataSetApplyTask");
//Create, store & execute apply Task
DataSetApplyTask applyTask = m_dsApplyFactory.create(
        " JDM_APPLY_PDS ", "JDM_TREE_MODEL", " JDM_APPLY_SETTINGS ",
        "JDM_APPLY_OUTPUT_TABLE");
//Save ApplyTask object
dmeConn.saveObject("JDM_APPLY_TASK", applyTask, true);
//Execute test task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_APPLY_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
```

Using a Cost Matrix

The class `javax.datamining.supervised.classification.CostMatrix` is used to represent the costs of the false positive and false negative predictions. It is used for classification problems to specify the costs associated with the false predictions.

In the Oracle Data Mining Java API, cost matrix is supported in apply and test operations for all classification models. For the decision tree algorithm, a cost matrix can be specified at build time. For more information about cost matrix, see *Oracle Data Mining Concepts*.

Example 7–9 illustrates how to create a cost matrix object where the target has two classes: `YES` (1) and `NO` (0). Suppose a positive (`YES`) response to the promotion generates $2 and the cost of the promotion is $1. Then the cost of misclassifying a positive responder is $2. The cost of misclassifying a non-responder is $1.

Example 7–9 Creating a Cost Matrix

```
//Create category set factory & cost matrix factory
CategorySetFactory catSetFactory = (CategorySetFactory)m_dmeConn.getFactory(
      "javax.datamining.data.CategorySet" );
CostMatrixFactory costMatrixFactory = (CostMatrixFactory)m_dmeConn.getFactory(
      "javax.datamining.supervised.classification.CostMatrix");
//Create categorySet
CategorySet catSet = m_catSetFactory.create(AttributeDataType.integerType);
//Add category values
catSet.addCategory(new Integer(0), CategoryProperty.valid);
catSet.addCategory(new Integer(1), CategoryProperty.valid);
//create cost matrix
CostMatrix costMatrix = m_costMatrixFactory.create(catSet);
costMatrix.setCellValue(new Integer(0), new Integer(0), 0);
costMatrix.setCellValue (new Integer(1), new Integer(1), 0);
costMatrix.setCellValue (new Integer(0), new Integer(1), 2);
costMatrix.setCellValue (new Integer(1), new Integer(0), 1);
//Save cost matrix in the DME
dmeConn.saveObject("JDM_COST_MATRIX", costMatrix);
```

Using Prior Probabilities

Prior probabilities are used for classification problems if the actual data has a different distribution for target values than the data provided for the model build. A user can specify the prior probabilities in the classification function settings, using `setPriorProbabilitiesMap`. For more information about prior probabilities, see *Oracle Data Mining Concepts*.

Note: Priors are not supported with decision trees.

Example 7–10 illustrates how to create a `PriorProbabilities` object, when the target has two classes: `YES` (1) and `NO` (0), and probability of `YES` is 0.05, probability of `NO` is 0.95.

Example 7–10 Creating Prior Probabilities

```
//Set target prior probabilities
Map priorMap = new HashMap();
priorMap.put(new Double(0), new Double(0.7));
priorMap.put(new Double(1), new Double(0.3));
buildSettings.setPriorProbabilitiesMap("affinity_card", priorMap);
```

Embedded Transformations

In 11.1, OJDM supports embedding transformations with the model metadata. When the transformations are embedded with the model, they are implicitly applied to apply and test datasets. For example, user can embed a transformation that recodes the response attributes value representation from 1/0 to Yes/No; model uses this transformation when applying the model to the new data.

Users can specify these transformations as SQL expressions or can use the OJDM transformations discussed in Section 2.13 and build a transformation sequence.

In this section, the first example discusses the simple expression transformation using the oracle.dmt.jdm.transform.OraExpressionTransform class specified as input for model build.

The second example illustrates how to build a sequence of complex transformations and persist them as a transformation sequence and embed them into the model.

Embed Single-Expression Transformations

Using `OraTransformationFactory` user can create transformation objects such as `OraTransformationSequence`, `OraExpressionTransform`, `OraBinningTransform`, `OraNormalizationTransform` and `OraClippingTransform`.

In Example 7–11, we create an expression transform that defines a simple log transformation for age attribute, recode transformation for affinity_card attribute and explicit exclusion of original age attribute from the model build. The code illustrates using OJDM API how one can embed these simple SQL expression transformations with the model.

Example 7–11 Simple Expression Transformation

```
//Create OraTransformationFactory
OraTransformationFactory m_xformFactory = (OraTransformationFactory)m_
dmeConn.getFactory(
      "oracle.dmt.jdm.transform.OraTransformation" );
//Create OraExpressionTransform from the transformation factory
OraExpressionTransform exprXform = m_xformFactory.createExpressionTransform();
  //1) Specify log transformation of age attribute and create a new attribute call
log_age
  //   that can be used for mining
  exprXform.addAttributeExpression("log_age", //Expression output attribute name
                                  "log(10, age) as log_age", //Expression
                                  "power(10, log_age)" //Reverse expression
                                  );
  //2) Recode 1/0 values of the affinity card attribute with the yes/no values and
replace
  //   existing attribute with the new recoded attribute
  exprXform.addAttributeExpression("affinity_card", //Expression output attribute
name
                                  "CASE WHEN AFFINITY_CARD = 1 THEN 'YES' ELSE
'NO' END ",
                                  null //No explicit reverse expression
                                  );
  //3) Exclude age attribute from mining
  exprXform.addAttributeExpression("age", //Expression output attribute name
                                  null, //Specify expression as null
                                //to exclude attribute from mining
                                  null
                                  );
//Create transformation sequence object using expression transformation
OraTransformationSequence xformSeq = m_xformFactory.createTransformationSequence(
        "MINING_DATA_BUILD_V", //Input table
        exprXform, //Expressions to be defined
        null //Output transformed view is specified as null as we are trying to
```

```
              //embed the transformations to the model
      );
//Save transformation sequence object
m_dmeConn.saveObject("simpleExprXForm_jdm", xformSeq, true);
//Create build Task  with transformation sequence
BuildTask buildTask = m_buildFactory.create(
                     "inputPDS", //Build data specification
                     "inputBuildSettings", //Mining function settings name
                     "outputModel" //Mining model name
                     );
//Specify transformation sequence as one of the input to enable embedding
//of the transformations defined in the sequence with the model
//In this example only expression transformations are specified
((OraBuildTask)buildTask).setTransformationSequenceName("simpleExprXForm_jdm");
//Save and execute the build task
...
//After successful model build specified transformations are embedded with the
model
//User can retrieve the transformation details that are embedded with the model by
calling
//the following function in OraModel
OraExpressionTransform modelExmbeededTransforms =
                     ((OraModel)model). GetModelTransformations();
```

Embed Complex Sequence of Transformations

In the previous example, we explored how to embed simple SQL expression transformations for trivial business transformations. In this section we will detail how a complex transformation sequence can be built using OJDM and embed these with the model.

OJDM 10.2 provides typical mining related individual transformations such as binning, normalization and outlier treatment (clipping). In 10.2 users have to maintain these transformations outside the modeling process and do the consistent transformations for the build, apply and test datasets outside the mining operations. This requires significant additional coding and maintenance of the transformation related objects by the end-user applications.

With the model embedded transformations capability, users can embed even complex transformation sequences such as first add business transformations and new attributes using the expression transforms (as discussed in the previous example), second treat outliers with the user specified clipping definitions and lastly normalize the data with the user specified normalization technique.

In OJDM new `OraTransformationSequence` object supports ability to specify sequence of transformations and convert these transformations into per attribute SQL expressions to embed them to the model. Example 7–12 illustrates using OJDM API how one can build a transformation sequence which uses the expression transform created in the previous example and extends it with the outlier and normalization data mining transformations and embed these complex transformation sequence with the model.

Example 7–12 Complex Sequence Transformations

```
//Create a list of transformations to be performed on the input mining data
List xformList = new ArrayList();
xfromList.add( exprXform );     //Expression transformation
xformList.add( clippingXform ); //Clipping transformation to treat outliers
xformList.add( normalizeXform );//Normalization transformation
//Create transformation sequence object using list of transformation
```

```
OraTransformationSequence xformSeq = m_xformFactory.createTransformationSequence(
        "MINING_DATA_BUILD_V", //Input table
        xformList, //List of transformations
        null //Output transformed view is specified as null as we are trying to
             //embed the transformations to the model
      );
//Save transformation sequence object
m_dmeConn.saveObject("complexXFormSeq_jdm", xformSeq, true);
//Create transformation task with the transformation sequence
OraTransformationTaskFactory m_xformTaskFactory =
       (OraTransformationTaskFactory)m_dmeConn.getFactory(
                   "oracle.dmt.jdm.task.OraTransformationTask");
OraTransformationTask xformTask =  m_xformTaskFactory .create(
                 "complexXFormSeq_jdm",,
                 false //boolean flag useTransformDefinitionTables
                 );
//Save and execute transformation task to populate transformation sequence with
the
//SQL expressions necessary before embedding them to the build task
. . . .
//Create build Task with transformation sequence
. . . .
((OraBuildTask)buildTask).setTransformationSequenceName("complexXFormSeq_jdm ");
//Save and execute the build task with the embedded transformations
...
```

Note that in both the examples we specified the output view of the transformation sequence as null to avoid creation of the database view that includes the transformations in the sequence. However, one can specify the view name to create a database view with the transformations and use this view as input to the model build to maintain the transformations outside the modeling process. OJDM API provides flexibility for the applications to choose the approach that best fits the need.

Using Predictive Analytics Tasks: Predict, Explain, and Profile

OJDM has `oracle.dmt.jdm.task.OraPredictTask`, `OraExplainTask` and `OraProfileTask` for data mining novice users to get predictions, to explain attributes importance and to discover profiles from the data.

Using `OraPredictTask` predictions are computed by just specifying the data location and the target column. This task learns from the known values in the target column and other columns in the input table and fills the unknown values in the target column with the predictions. This task hides all the data mining process done inside the database and produces the predictions and accuracy of the predictions.

Using `OraExplainTask` attributes ranking/importance with respect to an explain column. By just specifying the data location and explain column this task produces the attribute ranking table.

Using `OraProfileTask` profiles are discovered from the data for a given target attribute. For example, to find the profiles of the customers who respond to a product promotion, give the customers dataset with the customer attributes and promotion response attribute to the profile task. Profile task outputs a table with the profile definitions that applications can display to the users.

Both the tasks do automated data preparation where needed.

Example 7–13 illustrates how to execute predict, explain, and profile tasks.

Example 7–13 Predictive Analytics

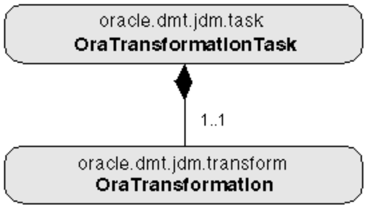
```
//Get Predictive Analytics Task Factory object
   OraPredictiveAnalyticsTaskFactory  m_paFactory =
     (OraPredictiveAnalyticsTaskFactory)m_dmeConn.getFactory(
          "oracle.dmt.jdm.task.OraPredictiveAnalyticsTask");
//Predict task
   //Create predict task object
   OraPredictTask predictTask = m_paFactory.createPredictTask (
                     "MINING_DATA_BUILD_V", //Input table
                     "cust_id", //Case id column
                     "affinity_card", //target column
                     "JDM_PREDICTION_RESULTS"); //prediction output table
   //Save predict task object
   dmeConn.saveObject("JDM_PREDICT_TASK", predictTask, true);
   //Execute test task asynchronously in the database
   ExecutionHandle execHandle1 = dmeConn.execute("JDM_PREDICT_TASK");
   //Wait for completion of the task
 ExecutionStatus execStatus1 = execHandle1.waitForCompletion(Integer.MAX_VALUE);
//Explain task
   //Create explain task object
   OraExplainTask explainTask = m_paFactory.createExplainTask (
                     "MINING_DATA_BUILD_V", //Input table
                     "affinity_card", //explain column
                     "JDM_EXPLAIN_RESULTS"); //explain output table
   //Save predict task object
   dmeConn.saveObject("JDM_EXPLAIN_TASK", explainTask, true);
   //Execute test task asynchronously in the database
   ExecutionHandle execHandle2 = dmeConn.execute("JDM_ EXPLAIN_TASK");
   //Wait for completion of the task
 ExecutionStatus execStatus2 = execHandle2.waitForCompletion(Integer.MAX_VALUE);
//Profile task
   //Create profile task
    OraProfileTask profileTask = m_paFactory.createProfileTask(
                         "MINING_DATA_BUILD_V", //Input table
                         "affinity_card", //Target column
                                 "JDM_PROFILE_RESULTS); //Profile output table
   //Save predict task object
   dmeConn.saveObject("JDM_PROFILE_TASK", profileTask, true);
   //Execute test task asynchronously in the database
   ExecutionHandle execHandle3 = dmeConn.execute("JDM_PROFILE_TASK");
   //Wait for completion of the task
   ExecutionStatus execStatus3 = execHandle3.waitForCompletion(Integer.MAX_VALUE);
```

Preparing the Data

In the Oracle Data Mining Java API, data must be prepared before building, applying, or testing a model. The `oracle.dmt.jdm.task.OraTransformationTask` class supports common transformations used in data mining: binning, normalization, clipping, and text transformations. For more information about transformations, see *Oracle Data Mining Concepts*.

The class diagram in Figure 7–8 illustrates the `OraTransformationTask` and its relationship with other objects.

Figure 7–8 OraTransformationTask Class Diagram

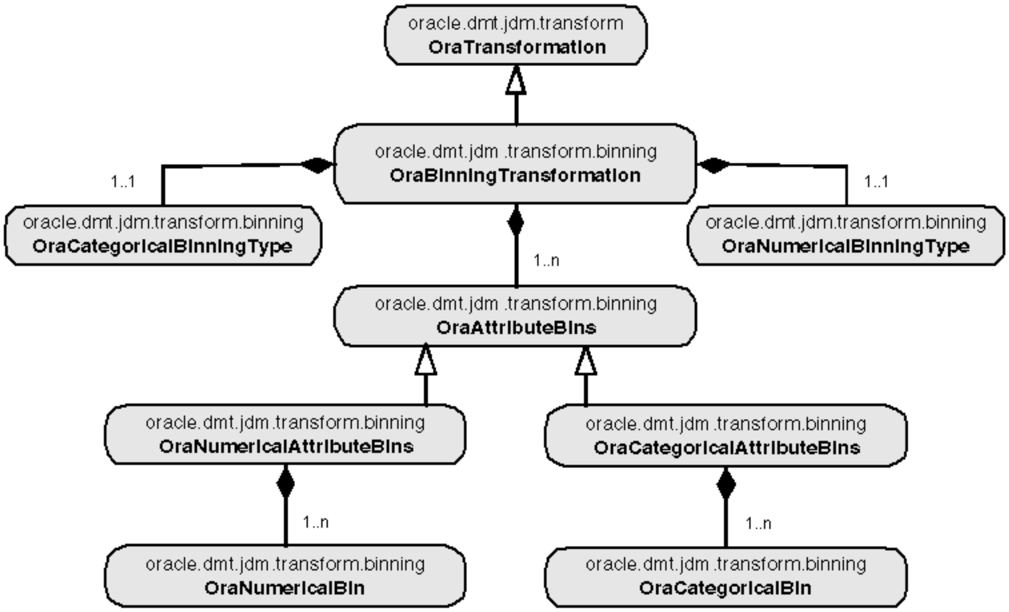


Using Binning/Discretization Transformation

Binning is the process of grouping related values together, thus reducing the number of distinct values for an attribute. Having fewer distinct values typically leads to a more compact model and one that builds faster, but it can also lead to some loss in accuracy.

The class diagram in Figure 7–9 illustrates the binning transformation classes.

Figure 7–9 OraBinningTransformation Class Diagram


oracle.dmt.jdm.transform
OraTransformation
oracle.dmt.jdm .transform.binning
OraBinningTransformation
1..1
oracle.dmt.jdm.transform.binning
OraCategoricalBinningType
1..1
oracle.dmt.jdm.transform.binning
OraNumericalBinningType
1..n
oracle.dmt.jdm .transform.binning
OraAttributeBins
oracle.dmt.jdm .transform.binning
OraNumericalAttributeBins
oracle.dmt.jdm .transform.binning
OraCategoricalAttributeBins
1..n
1..n
oracle.dmt.jdm .transform.binning
OraNumericalBin
oracle.dmt.jdm .transform.binning
OraCategoricalBin

Here, `OraBinningTransformation` contains all the settings required for binning. The Oracle Data Mining Java API supports top-n, custom binning for categorical attributes, and equi-width, quantile and custom binning for numerical attributes. After running the binning transformations, it creates a transformed table and bin boundary tables in the user's schema. The user can specify the bin boundary table names, or the system will generate the names for the bin boundary tables. This facilitates the reusing of the bin boundary tables that are created for binning build data for apply and test data.

The following code illustrates the binning operation on the view `MINING_BUILD_DATA_V`

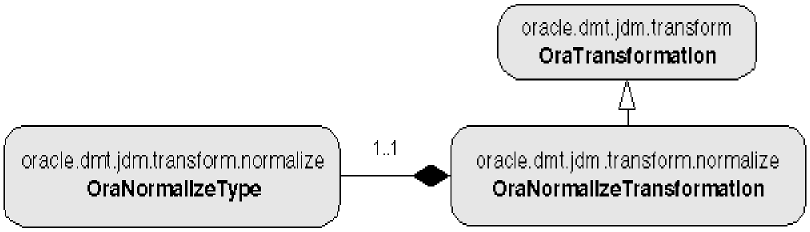
```
//Create binning transformation instance
OraBinningTransformFactory binXformFactory =
   (OraBinningTransformFactory)dmeConn.getFactory(
      "oracle.dmt.jdm.transform.binning.OraBinningTransform");
OraBinningTransform binTransform = m_binXformFactory.create(
      "MINING_DATA_BUILD_V", // name of the input data set
      "BINNED_DATA_BUILD_V", // name of the transformation result
      true); // result of the transformation is a view
// Specify the number of numeric bins
binTransform.setNumberOfBinsForNumerical(10);
// Specify the number of categoric bins
binTransform.setNumberOfBinsForCategorical(8);
// Specify the list of excluded attributes
String[] excludedList = new String[]{"CUST_ID", "CUST_GENDER"};
binTransform.setExcludeColumnList(excludedList);
// Specify the type of numeric binning: equal-width or quantile
       ( default is quantile )
binTransform.setNumericalBinningType(binningType);
// Specify the type of categorical binning as Top-N: by default it is none
binTransform.setCategoricalBinningType(OraCategoricalBinningType.top_n);
//Create transformation task
OraTransformationTask xformTask = m_xformTaskFactory.create(binTransform);
//Save transformation task object
dmeConn.saveObject("JDM_BINNING_TASK", xformTask, true);
//Execute transformation task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_ BINNING _TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
```

Using Normalization Transformation

Normalizing converts individual attribute values in such a way that all attribute values lie in the same range. Normally, values are converted to be in the range 0.0 to 1.0 or the range -1 to +1. Normalization ensures that attributes do not receive artificial weighting caused by differences in the ranges that they span.

The class diagram in Figure 7–10 illustrates the normalization transformation classes.

Figure 7–10 OraNormalizeTransformation Class Diagram


oracle.dmt.jdm.transform
OraTransformation
oracle.dmt.jdm.transform.normalize
OraNormalizeType
1..1
oracle.dmt.jdm.transform.normalize
OraNormalizeTransformation

Here, `OraNormalizeTransformation` contains all the settings required for normalization. The Oracle Data Mining Java API supports z-Score, min-max, and linear scale normalizations. Normalization is required for SVM, NMF, and *k*-Means algorithms.

The following code illustrates normalization on the view `MINING_BUILD_DATA_V`.

```
//Create OraNormalizationFactory
OraNormalizeTransformFactory normalizeXformFactory =
  (OraNormalizeTransformFactory)m_dmeConn.getFactory(
      "oracle.dmt.jdm.transform.normalize.OraNormalizeTransform");
//Create OraNormalization
OraNormalizeTransform normalizeTransform = m_normalizeXformFactory.create(
      "MINING_DATA_BUILD_V", // name of the input data set
      "NORMALIZED_DATA_BUILD_V", // name of the transformation result
      true, // result of the transformation is a view
      OraNormalizeType.z_Score, //Normalize type
      new Integer(6) ); //Rounding number
```

```
// Specify the list of excluded attributes
String[] excludedList = new String[]{"CUST_ID", "CUST_GENDER"};
normalizeTransform.setExcludeColumnList(excludedList);
//Create transformation task
OraTransformationTask xformTask = m_xformTaskFactory.create(normalizeTransform);
//Save transformation task object
dmeConn.saveObject("JDM_NORMALIZE_TASK", xformTask, true);
//Execute transformation task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_NORMALIZE_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
```

Using Clipping Transformation

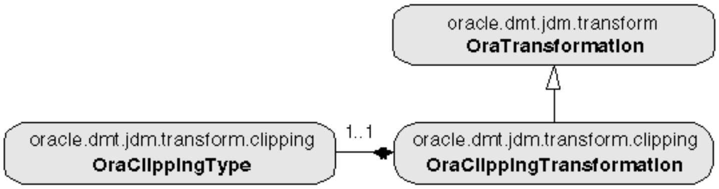
Some computations on attribute values can be significantly affected by extreme values. One approach to achieving a more robust computation is to either winsorize or trim the data using clipping transformations.

Winsorizing involves setting the tail values of a particular attribute to some specified value. For example, for a 90% winsorization, the bottom 5% are set equal to the minimum value in the 6th percentile, while the upper 5% are set equal to the value corresponding to the maximum value in the 95th percentile.

Trimming "removes" the tails in the sense that trimmed values are ignored in further values. This is achieved by setting the tails to NULL.

The class diagram in Figure 7–11 illustrates the clipping transformation classes.

Figure 7–11 OraClippingTransformation Class Diagram


oracle.dmt.jdm.transform
OraTransformation
oracle.dmt.jdm.transform.clipping
OraClippingType
1..1
oracle.dmt.jdm.transform.clipping
OraClippingTransformation

Here, `OraClippingTransformation` contains all the settings required for clipping. The Oracle Data Mining Java API supports winsorize and trim types of clipping.

The following code illustrates clipping on the view `MINING_BUILD_DATA_V`.

```
//Create OraClippingTransformFactory
OraClippingTransformFactory clipXformFactory =
  (OraClippingTransformFactory)dmeConn.getFactory(
      "oracle.dmt.jdm.transform.clipping.OraClippingTransform");
//Create OraClippingTransform
OraClippingTransform clipTransform = clipXformFactory.create(
      "MINING_DATA_BUILD_V", // name of the input data set
      "WINSORISED_DATA_BUILD_V", // name of the transformation result
      true );// result of the transformation is a view
//Specify the list of excluded attributes
String[] excludedList = new String[]{"CUST_ID", "CUST_GENDER"};
clipTransform.setExcludeColumnList(excludedList);
//Specify the type of clipping
clipTransform.setClippingType(OraClippingType.winsorize);
// Specify the tail fraction as 3% of values on both ends
clipTransform.setTailFraction(0.03);
//Create and save transformation task
OraTransformationTask xformTask = xformTaskFactory.create(clipTransform);
//Save transformation task object
```

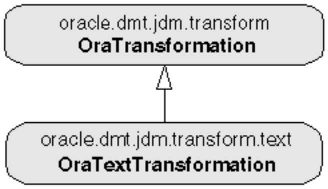
```
dmeConn.saveObject("JDM_CLIPPING_TASK", xformTask, true);
//Execute transformation task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_CLIPPING_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion(Integer.MAX_VALUE);
```

Using Text Transformation

Text columns need to be transformed to nested table structure to do the mining on text columns. This transformation converts the text columns to nested table columns. A features table is created by text transformation. A model build text data column features table must be used for apply and test tasks to get the correct results.

The class diagram in Figure 7–12 illustrates the text transformation classes.

Figure 7–12 Text Transformation Class Diagram



Here, `OraTextTransformation` is used to specify the text columns and the feature tables associated with the text columns.

The following code illustrates clipping on the table `MINING_BUILD_TEXT`.

```
//Create OraTextTransformFactory
OraTextTransformFactory textXformFactory = dmeConn.getFactory(
      "oracle.dmt.jdm.transform.text.OraTextTransform");
//Create OraTextTransform
OraTextTransform txtXform = (OraTextTransformImpl)textXformFactory.create(
      "MINING_BUILD_TEXT", // name of the input data set
      "NESTED_TABLE_BUILD_TEXT ", // name of the transformation result
      "CUST_ID", //Case id column
      new String[] { "COMMENTS" } ); //Text column names
      );
//Create transformation task
OraTransformationTask xformTask = m_xformTaskFactory.create(txtXform);
//Save transformation task object
dmeConn.saveObject("JDM_TEXTXFORM_TASK", xformTask, true);
//Execute transformation task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute("JDM_TEXTXFORM_TASK");
//Wait for completion of the task
ExecutionStatus execStatus = execHandle.waitForCompletion
     (Integer.MAX_VALUE);
```

Index