62 DBMS_DATA_MINING

The `DBMS_DATA_MINING` package is the application programming interface for creating, evaluating, and querying Oracle Machine Learning for SQL models.

In Oracle Database Release 21c, Oracle Data Mining has been rebranded to Oracle Machine Learning for SQL (Oracle Machine Learning for SQL). The PL/SQL package name, however, has not changed and remains `DBMS_DATA_MINING`.

This chapter contains the following topics:

See Also:

- *Oracle Machine Learning for SQL Concepts*
- *Oracle Machine Learning for SQL User's Guide*
- DBMS_DATA_MINING_TRANSFORM
- DBMS_PREDICTIVE_ANALYTICS

DBMS_DATA_MINING Overview

Oracle Machine Learning for SQL supports both supervised and unsupervised machine learning. Supervised machine learning predicts a target value based on historical data. Unsupervised machine learning discovers natural groupings and does not use a target. You can use Oracle Machine Learning for SQL procedures on structured data and unstructured text.

Supervised machine learning techniques include:

- Classification
- Regression
- Feature Selection (Attribute Importance)
- Time Series

Unsupervised machine learning techniques include:

- Clustering
- Association
- Feature Extraction
- Anomaly Detection

The steps you use to build and apply a machine learning model depend on the machine learning technique and the algorithm being used. The algorithms supported by Oracle Machine Learning for SQL are listed in the following table.

Table 62-1 Oracle Machine Learning for SQL Algorithms

Algorithm	Abbreviation	Function
Apriori	AR	Association
CUR Matrix Decomposition	CUR	Attribute importance
Decision Tree	DT	Classification
Expectation Maximization	EM	Clustering
Explicit Semantic Analysis	ESA	Feature extraction, classification
Exponential Smoothing	ESM	Time series
Generalized Linear Models	GLM	Classification, regression
k-Means	KM	Clustering
Minimum Descriptor Length	MDL	Attribute importance
Multivariate State Estimation Technique - Sequential Probability Ratio Test	MSET-SPRT	Anomaly detection, classification
Naive Bayes	NB	Classification
Neural Network	NN	Classification, regression
Non-Negative Matrix Factorization	NMF	Feature extraction
Orthogonal Partitioning Clustering	O-Cluster	Clustering
Random Forest	RF	Classification
Singular Value Decomposition and Principal Component Analysis	SVD and PCA	Feature extraction
Support Vector Machine	SVM	Classification, regression, anomaly detection
XGBoost	XGBoost	Classification, regression

Oracle Machine Learning for SQL supports more than one algorithm for the classification, regression, clustering, and feature extraction machine learning techniques. Each of these machine learning techniques has a default algorithm, as shown in the following table.

Table 62-2 Oracle Machine Learning for SQL Default Algorithms

Mining Function	Default Algorithm
Classification	Naive Bayes
Clustering	*k*-Means
Feature Extraction	Non-Negative Matrix Factorization
Feature Selection	Minimum Descriptor Length

Table 62-2 (Cont.) Oracle Machine Learning for SQL Default Algorithms

Mining Function	Default Algorithm
Regression	Support Vector Machine
Time Series	Exponential Smoothing

DBMS_DATA_MINING Security Model

The `DBMS_DATA_MINING` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The `DBMS_DATA_MINING` package exposes APIs that are leveraged by the Oracle Machine Learning for SQL. Users who wish to create machine learning models in their own schema require the `CREATE MINING MODEL` system privilege. Users who wish to create machine learning models in other schemas require the `CREATE ANY MINING MODEL` system privilege.

Users have full control over managing models that exist within their own schema. Additional system privileges necessary for managing machine learning models in other schemas include `ALTER ANY MINING MODEL`, `DROP ANY MINING MODEL`, `SELECT ANY MINING MODEL`, `COMMENT ANY MINING MODEL`, and `AUDIT ANY`.

Individual object privileges on machine learning models, `ALTER MINING MODEL` and `SELECT MINING MODEL`, can be used to selectively grant privileges on a model to a different user.

> **See Also:**
>
> *Oracle Data Mining User's Guide* for more information about the security features of Oracle Machine Learning for SQL

DBMS_DATA_MINING — Machine Learning Functions

A machine learning **function** refers to the methods for solving a given class of machine learning problems.

The machine learning function must be specified when a model is created. You specify a machine learning function with the `mining_function` parameter of the CREATE_MODEL Procedure or the CREATE_MODEL2 Procedure.

Table 62-3 Machine Learning Functions

Value	Description
`ASSOCIATION`	Association is a descriptive machine learning function. An association model identifies relationships and the probability of their occurrence within a data set. Association models use the Apriori algorithm.

Table 62-3 (Cont.) Machine Learning Functions

Value	Description
`ATTRIBUTE_IMPORTANCE`	Attribute importance is a predictive machine learning function, also known as feature selection. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome. Attribute importance models can use Minimum Description Length (MDL) or CUR Matrix Decomposition. MDL is the default.
`CLASSIFICATION`	Classification is a predictive machine learning function. A classification model uses historical data to predict a categorical target. Classification models can use: Decision Tree, logistic regression, Multivariate State Estimation Technique - Sequential Probability Ratio Test, Naive Bayes, Support Vector Machine (SVM), or XGBoost. The default is Naive Bayes. The classification function can also be used for **anomaly detection**. For anomaly detection, you can use the Multivariate State Estimation Technique - Sequential Probability Ratio Test algorithm or the SVM algorithm with a null target (One-Class SVM), or the EM algorithm with a null target (EM Anomaly).
`CLUSTERING`	Clustering is a descriptive machine learning function. A clustering model identifies natural groupings within a data set. Clustering models can use *k*-Means, O-Cluster, or Expectation Maximization. The default is *k*-Means.
`FEATURE_EXTRACTION`	Feature extraction is a descriptive machine learning function. A feature extraction model creates an optimized data set on which to base a model. Feature extraction models can use Explicit Semantic Analysis, Non-Negative Matrix Factorization, Singular Value Decomposition, or Principal Component Analysis. Non-Negative Matrix Factorization is the default.
`REGRESSION`	Regression is a predictive machine learning function. A regression model uses historical data to predict a numerical target. Regression models can use linear regression, Support Vector Machine, or XGBoost. The default is Support Vector Machine.
`TIME_SERIES`	Time series is a predictive machine learning function. A time series model forecasts the future values of a time-ordered series of historical numeric data over a user-specified time window. Time series models use the Exponential Smoothing algorithm.

See Also:

Oracle Machine Learning for SQL Concepts for more information about mining functions

DBMS_DATA_MINING — Model Settings

Oracle Machine Learning for SQL uses settings to specify the algorithm and other characteristics of a model. Some settings are general, some are specific to a machine learning function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, then you must create a settings table. The settings table must have the column names and data types shown in the following table.

Table 62-4 Required Columns in the Model Settings Table

Column Name	Data Type
`SETTING_NAME`	`VARCHAR2(30)`
`SETTING_VALUE`	`VARCHAR2(4000)`

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the CREATE_MODEL Procedure. You can also provide these settings through the CREATE_MODEL2 Procedure.

The settings used by a model can be found by querying the data dictionary view `ALL_MINING_MODEL_SETTINGS`. This view displays the model settings used by the machine learning models to which you have access. All of the default and user-specified setting values are included in the view.

See Also:

- `ALL_MINING_MODEL_SETTINGS` in *Oracle Database Reference*
- *Oracle Machine Learning for SQL User's Guide* for information about specifying model settings

DBMS_DATA_MINING — Algorithm Names

The `ALGO_NAME` setting specifies the model algorithm.

The values for the `ALGO_NAME` setting are listed in the following table.

Table 62-5 Algorithm Names

ALGO_NAME Value	Description	Machine Learning Function
`ALGO_AI_MDL`	Minimum Description Length	Attribute importance
`ALGO_APRIORI_ASSOCIATION_RULES`	Apriori	Association rules
`ALGO_CUR_DECOMPOSITION`	CUR Matrix Decomposition	Attribute importance
`ALGO_DECISION_TREE`	Decision Tree	Classification
`ALGO_EXPECTATION_MAXIMIZATION`	Expectation Maximization	Clustering, Classification

Table 62-5 (Cont.) Algorithm Names

ALGO_NAME Value	Description	Machine Learning Function
ALGO_EXPLICIT_SEMANTIC_ANALYS	Explicit Semantic Analysis	Feature extraction Classification
ALGO_EXPONENTIAL_SMOOTHING	Exponential Smoothing	Time series
ALGO_EXTENSIBLE_LANG	Language used for extensible algorithm	All mining functions supported
ALGO_GENERALIZED_LINEAR_MODEL	Generalized Linear Model	Classification, regression; also feature selection and generation
ALGO_KMEANS	Enhanced *k*-Means	Clustering
ALGO_MSET_SPRT	Multivariate State Estimation Technique - Sequential Probability Ratio Test	Classification
ALGO_NAIVE_BAYES	Naive Bayes	Classification
ALGO_NEURAL_NETWORK	Neural Network	Classification
ALGO_NONNEGATIVE_MATRIX_FACTOR	Non-Negative Matrix Factorization	Feature extraction
ALGO_O_CLUSTER	O-Cluster	Clustering
ALGO_RANDOM_FOREST	Random Forest	Classification
ALGO_SINGULAR_VALUE_DECOMP	Singular Value Decomposition	Feature extraction
ALGO_SUPPORT_VECTOR_MACHINES	Support Vector Machine	Classification and regression
ALGO_XGBOOST	XGBoost	Classification and regression

See Also:

Oracle Machine Learning for SQL Concepts for information about algorithms

DBMS_DATA_MINING — Automatic Data Preparation

Oracle Machine Learning for SQL supports fully Automatic Data Preparation (ADP), user-directed general data preparation, and user-specified embedded data preparation. The `PREP_*` settings enable the user to request fully automated or user-directed general data preparation. By default, fully Automatic Data Preparation (`PREP_AUTO_ON`) is enabled.

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. Instead of fully ADP, the user can request that the data be shifted and/or scaled with the `PREP_SCALE*` and `PREP_SHIFT*` settings. The transformation instructions are stored with the model and reused whenever the model is applied. The model settings can be viewed in `USER_MINING_MODEL_SETTINGS`.

You can choose to supplement Automatic Data Preparations by specifying additional transformations in the `xform_list` parameter when you build the model. See "CREATE_MODEL Procedure" and "CREATE_MODEL2 Procedure".

If you do not use ADP *and* do not specify transformations in the `xform_list` parameter to `CREATE_MODEL`, you must implement your own transformations separately in the build, test, and scoring data. You must take special care to implement the exact same transformations in each data set.

If you do not use ADP, but you *do* specify transformations in the `xform_list` parameter to `CREATE_MODEL`, OML4SQL embeds the transformation definitions in the model and prepares the test and scoring data to match the build data.

The values for the `PREP_*` setting are described in the following table.

Table 62-6 PREP_* Setting

Setting Name	Setting Value	Description
`PREP_AUTO`	• `PREP_AUTO_ON` • `PREP_AUTO_OFF`	This setting enables fully automated data preparation. The default is `PREP_AUTO_ON`.
`PREP_SCALE_2DNUM`	• `PREP_SCALE_STDDEV` • `PREP_SCALE_RANGE`	This setting enables scaling data preparation for two-dimensional numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. The following are the possible values: • `PREP_SCALE_STDDEV`: A request to divide the column values by the standard deviation of the column and is often provided together with `PREP_SHIFT_MEAN` to yield z-score normalization. • `PREP_SCALE_RANGE`: A request to divide the column values by the range of values and is often provided together with `PREP_SHIFT_MIN` to yield a range of [0,1].
`PREP_SCALE_NNUM`	`PREP_SCALE_MAXABS`	This setting enables scaling data preparation for nested numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. If specified, then the valid value for this setting is `PREP_SCALE_MAXABS`, which yields data in the range of [-1,1].
`PREP_SHIFT_2DNUM`	• `PREP_SHIFT_MEAN` • `PREP_SHIFT_MIN`	This setting enables centering data preparation for two-dimensional numeric columns. `PREP_AUTO` must be `OFF` for this setting to take effect. The following are the possible values: • `PREP_SHIFT_MEAN`: Results in subtracting the average of the column from each value. • `PREP_SHIFT_MIN`: Results in subtracting the minimum of the column from each value. .

> **See Also:**
>
> *Oracle® Machine Learning for SQL* for information about data transformations

DBMS_DATA_MINING — Machine Learning Function Settings

The settings described in this table apply to a machine learning function.

Table 62-7 Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Association	`ASSO_MAX_RULE_LENGTH`	An integer between 2 to 20, inclusive, represented as a character string	Sets the upper limit on the length of association rules. Useful in managing the compute time. Shorter rules (fewer antecedents) take less memory and compute time. Expression: `TO_CHAR(3)` Default is `4`.
Association	`ASSO_MIN_CONFIDENCE`	A floating point number between 0 and 1, inclusive, expressed as a character string	Defines the minimum confidence threshold for association rules. This parameter reduces the number of rules generated, focusing only on those that meet the minimum confidence specification. It reduces model size. Expression: `TO_CHAR(0.4)` Default is `0.1`.
Association	`ASSO_MIN_SUPPORT`	A floating point number between 0 and 1, inclusive, expressed as a character string	Specifies the minimum support threshold for association rules. Expression: `TO_CHAR(0.2)` Default is `0.1`.
Association	`ASSO_MIN_SUPPORT_INT`	a positive integer	Determines the minimum absolute support each rule must meet, expressed as an integer. Sets a concrete baseline for the number of occurrences required for an itemset to be considered, ensuring rules are based on sufficiently frequent patterns. The default is `1`.
Association	`ASSO_MIN_REV_CONFIDENCE`	A floating point number between 0 and 1, inclusive, expressed as a character string.	Establishes the minimum reverse confidence for each association rule. This setting filters rules to ensure that they are significant not just in the context of the antecedent but also regarding the consequent, enhancing the relevance of the rule. The Reverse Confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs. The value is real number between 0 and 1. Expression: `TO_CHAR(0.45)` The default is `0`.

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Association	`ASSO_IN_RULES`	`NULL`	Specifies a list of items that must be included in each association rule. It accepts a comma-separated string of items; at least one of these must appear in every reported rule, either as an antecedent or a consequent. This parameter is useful for focusing the analysis on rules that involve specific items of interest, thereby tailoring the association rules to specific analytical needs or hypotheses. If not set, the filtering is not applied by default. For example, `INSERT INTO sett_tab (setting_name, setting_value) VALUES (dbms_data_mining.asso_in_rules, '''a'',''b''');`
Association	`ASSO_EX_RULES`	`NULL`	Defines a list of items to be excluded from each association rule. Accepts a comma-separated string listing items that should not appear in any reported association rules. Essential for omitting specific items from rule generation, which can be particularly useful when certain items are known to be irrelevant or misleading in the context of the analysis. The default is `NULL`. For example, `INSERT INTO sett_tab (setting_name, setting_value) VALUES (dbms_data_mining.asso_ex_rules, '''a'',''b''');`
Association	`ASSO_ANT_IN_RULES`	`NULL`	Determines inclusion criteria for the antecedent part of each association rule. Accepts a comma-separated list of items where at least one must appear in the antecedent of each rule. Targets specific items to always be considered as potential causes in the rules, refining the focus of the analysis. The default is `NULL`. For example, `INSERT INTO sett_tab (setting_name, setting_value) VALUES (dbms_data_mining.asso_ant_in_rules, '''a'',''b''');`

Table 62-7 (Cont.) Machine Learning Function Settings

<table>
<tr><th>Machine Learning Function</th><th>Setting Name</th><th>Setting Value</th><th>Description</th></tr>
<tr><td>Association</td><td><code>ASSO_ANT_EX_RULES</code></td><td><code>NULL</code></td><td>Sets exclusion criteria for the antecedent in association rules. Comma-separated string listing items to be excluded from the antecedent of each rule. Prevents specified items from being considered as causes in the rules, avoiding irrelevant or known redundant combinations.

The default is <code>NULL</code>.

The following example illustrates how you can define the parameter when you are using the <code>CREATE_MODEL</code> procedure. You must create a table with setting name (a constant) and setting value and then use the <code>CREATE_MODEL</code> procedure.

<pre>INSERT INTO sett_tab (setting_name,
setting_value) VALUES

(dbms_data_mining.asso_ant_ex_rules,
'''a'',''b''');</pre>
The following example illustrates how you can define the parameter when you are using the <code>CREATE_MODEL2</code> procedure using string values.

<pre>%script
BEGIN
DBMS_DATA_MINING.DROP_MODEL('AR_SH_SAMP
LE');
EXCEPTION WHEN OTHERS THEN NULL; END;
/
DECLARE
    v_setlst
DBMS_DATA_MINING.SETTING_LIST;
BEGIN

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

v_setlst('PREP_AUTO')   :=
'ON';

v_setlst('ASSO_MIN_SUPPORT')   :=
'0.04';

v_setlst('ASSO_MIN_CONFIDENCE')   :=
'0.1';

v_setlst('ASSO_MAX_RULE_LENGTH')   :=
'2';</pre></td></tr>
</table>

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

			`v_setlst('ASSO_ANT_EX_RULES') := '''a'',''b''';`
			
			`v_setlst('ODMS_ITEM_ID_COLUMN_NAME'):= 'PROD_NAME';`
			
			`v_setlst('ASSO_AGGREGATES') := 'AMOUNT_SOLD';`
			
			`DBMS_DATA_MINING.CREATE_MODEL2(`
			`MODEL_NAME => 'AR_SH_SAMPLE',`
			`MINING_FUNCTION => 'ASSOCIATION',`
			`DATA_QUERY => 'select * from SALES_TRANS_CUST',`
			`SET_LIST => v_setlst,`
			`CASE_ID_COLUMN_NAME => 'CUST_ID');`
			`END;`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Association	`ASSO_CONS_IN_RULES`	`NULL`	Establishes inclusion criteria for the consequent part of each association rule. Specifies a list of items where at least one must appear in the consequent of each rule. Ensures that certain items are always considered as potential outcomes in the rules, focusing on specific effects of interest.
			The default is `NULL`.
			The following example illustrates how you can define the parameter when you are using the `CREATE_MODEL` procedure. You must create a table with setting name (a constant) and setting value and then use the `CREATE_MODEL` procedure.
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_cons_in_rules, '''a'',''b''');`
			The following example illustrates how you can define the parameter when you are using the `CREATE_MODEL2` procedure using string values.
			`%script`
			`BEGIN`
			`DBMS_DATA_MINING.DROP_MODEL('AR_SH_SAMPLE');`
			`EXCEPTION WHEN OTHERS THEN NULL; END;`
			`/`
			`DECLARE`
			`    v_setlst DBMS_DATA_MINING.SETTING_LIST;`
			`BEGIN`
			`v_setlst('ALGO_NAME') := 'ALGO_APRIORI_ASSOCIATION_RULES';`
			`v_setlst('PREP_AUTO') := 'ON';`
			`v_setlst('ASSO_MIN_SUPPORT') := '0.04';`
			`v_setlst('ASSO_MIN_CONFIDENCE') := '0.1';`
			`v_setlst('ASSO_MAX_RULE_LENGTH') := '2';`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

			`v_setlst('ASSO_CONS_IN_RULES')      := '''a'',''b''';`
			
			`v_setlst('ODMS_ITEM_ID_COLUMN_NAME'):= 'PROD_NAME';`
			
			`v_setlst('ASSO_AGGREGATES')         := 'AMOUNT_SOLD';`
			
			`    DBMS_DATA_MINING.CREATE_MODEL2(`
			`        MODEL_NAME        => 'AR_SH_SAMPLE',`
			`        MINING_FUNCTION   => 'ASSOCIATION',`
			`        DATA_QUERY        => 'select * from SALES_TRANS_CUST',`
			`        SET_LIST          => v_setlst,`
			`        CASE_ID_COLUMN_NAME => 'CUST_ID');`
			`END;`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Association	`ASSO_CONS_EX_RULES`	`NULL`	Sets exclusion criteria for the consequent in association rules. Comma-separated string indicating items to be excluded from the consequent of each rule. Omits specific items from being outcomes in the rules, removing non-relevant or misleading results.
			The excluding rule can be used to reduce the rules that must be stored, but the user may be required to build an extra model for running different including or excluding rules.
			The default is `NULL`.
			The following example illustrates how you can define the parameter when you are using the `CREATE_MODEL` procedure. You must create a table with setting name (a constant) and setting value and then use the `CREATE_MODEL` procedure.
			`INSERT INTO sett_tab (setting_name, setting_value) VALUES`
			`(dbms_data_mining.asso_cons_ex_rules, '''a'',''b''');`
			The following example illustrates how you can define the parameter when you are using the `CREATE_MODEL2` procedure using string values.
			`%script`
			`BEGIN`
			`DBMS_DATA_MINING.DROP_MODEL('AR_SH_SAMP LE');`
			`EXCEPTION WHEN OTHERS THEN NULL; END;`
			`/`
			`DECLARE`
			`    v_setlst DBMS_DATA_MINING.SETTING_LIST;`
			`BEGIN`
			`v_setlst('ALGO_NAME')         := 'ALGO_APRIORI_ASSOCIATION_RULES';`
			`v_setlst('PREP_AUTO')         := 'ON';`
			`v_setlst('ASSO_MIN_SUPPORT')  := '0.04';`
			`v_setlst('ASSO_MIN_CONFIDENCE')     := '0.1';`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

			`v_setlst('ASSO_MAX_RULE_LENGTH')     :=`
			`'2';`
			
			`v_setlst('ASSO_CONS_EX_RULES')       :=`
			`'''a'',''b''';`
			
			`v_setlst('ODMS_ITEM_ID_COLUMN_NAME'):=`
			`'PROD_NAME';`
			
			`v_setlst('ASSO_AGGREGATES')          :=`
			`'AMOUNT_SOLD';`
			
			`    DBMS_DATA_MINING.CREATE_MODEL2(`
			`        MODEL_NAME         =>`
			`'AR_SH_SAMPLE',`
			`        MINING_FUNCTION    =>`
			`'ASSOCIATION',`
			`        DATA_QUERY         => 'select *`
			`from SALES_TRANS_CUST',`
			`        SET_LIST           => v_setlst,`
			`        CASE_ID_COLUMN_NAME =>`
			`'CUST_ID');`
			`END;`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Association	`ASSO_AGGREGATES`	`NULL`	Defines columns for aggregation in association rules. Comma-separated list of column names for aggregation, limited to 10 columns. Aggregates additional data alongside items, providing more context to the association rules, but may increase memory and computational requirements.
			You can specify `ASSO_AGGREGATES` if `ODMS_ITEM_ID_COLUMN_NAME` is set indicating transactional input data. See DBMS_DATA_MINING - Global Settings. The data table must have valid column names such as `ITEM_ID` and `CASE_ID` which are derived from `ODMS_ITEM_ID_COLUMN_NAME` and `case_id_column_name` respectively. Numeric values are supported. `ITEM_VALUE` is not a mandatory value.
			The default is `NULL`.
			For example, if the following is your Transactional Data table:
			`CREATE OR REPLACE VIEW SALES_TRANS_CUST AS`
			`  SELECT DISTINCT CUST_ID, PROD_NAME, PROD_CATEGORY`
			`    FROM (SELECT A.CUST_ID, B.PROD_NAME, B.PROD_CATEGORY`
			`    FROM SH.SALES A, SH.PRODUCTS B`
			`    WHERE A.PROD_ID = B.PROD_ID AND`
			`      A.CUST_ID BETWEEN 100001 AND 104500);`
			Then, you can create your model similar to:
			`%script`
			`BEGIN`
			`DBMS_DATA_MINING.DROP_MODEL('AR_SH_SAMPLE');`
			`EXCEPTION WHEN OTHERS THEN NULL; END;`
			`/`
			`DECLARE`
			`    v_setlst DBMS_DATA_MINING.SETTING_LIST;`
			`BEGIN`
			`v_setlst('ALGO_NAME')               :=`

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

			`'ALGO_APRIORI_ASSOCIATION_RULES';`
			
			`v_setlst('PREP_AUTO') := 'ON';`
			
			`v_setlst('ASSO_MIN_SUPPORT') := '0.04';`
			
			`v_setlst('ASSO_MIN_CONFIDENCE') := '0.1';`
			
			`v_setlst('ASSO_MAX_RULE_LENGTH') := '2';`
			
			`v_setlst('ODMS_ITEM_ID_COLUMN_NAME'):= 'PROD_NAME';`
			
			`v_setlst('ASSO_AGGREGATES') := 'AMOUNT_SOLD';`
			
			`DBMS_DATA_MINING.CREATE_MODEL2(`
			`MODEL_NAME => 'AR_SH_SAMPLE',`
			`MINING_FUNCTION => 'ASSOCIATION',`
			`DATA_QUERY => 'select * from SALES_TRANS_CUST',`
			`SET_LIST => v_setlst,`
			`CASE_ID_COLUMN_NAME => 'CUST_ID');`
			`END;`
			
			The default is `NULL`.
			For each item, the user may supply several columns to aggregate. It requires more memory to buffer the extra data. Also, the performance impact can be seen because of the larger input data set and more operation.

Association	`ASSO_ABS_ERROR`	`0<ASSO_ABS_ERRORMAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE).`	Specifies the absolute error for the association rules sampling. Balances accuracy and computational efficiency in rule sampling; smaller values lead to larger samples and more precise results but increase computation time. Set a reasonable value for `ASSO_ABS_ERROR`, such as its default value, to avoid large sample size. The default value is `0.5` * `MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)`.

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Association	`ASSO_CONF_LEVEL`	`0` `ASSO_CONF_LEVEL 1`	Sets the confidence level for an association rules sample. A higher confidence level increases sample size. Any value between 0.9 and 1 is suitable. The default value is `0.95`.
Classification	`CLAS_COST_TABLE_NAME`	*table_name*	(Decision tree only) Names a user-created cost matrix table for model building. The cost matrix specifies misclassification costs. This parameter tailors decision tree models to prioritize certain types of misclassifications, enhancing model effectiveness in specific scenarios. Only decision tree models can use a cost matrix at build time. All classification algorithms can use a cost matrix at apply time. See "ADD_COST_MATRIX Procedure" for the column requirements. See *Oracle Machine Learning for SQL Concepts* for information about costs.
Classification	`CLAS_PRIORS_TABLE_NAME`	*table_name*	(Naive Bayes) Names a user-created table for storing prior probabilities. Adjusts for distribution differences between build and scoring data. Aligns model training more closely with real-world data distributions, improving prediction accuracy for Naive Bayes models. See *Oracle Machine Learning for SQL User's Guide* for the column requirements. See *Oracle Machine Learning for SQL Concepts* for additional information about priors.
Classification	`CLAS_WEIGHTS_TABLE_NAME`	*table_name*	(GLM and SVM only) Names a user-created table for storing weights for target values. Weights bias the model towards higher weighted classes. This parameter adjusts GLM and SVM models to focus on or balance between different classes, enhancing model performance for specific targets. See *Oracle Machine Learning for SQL User's Guide* for the column requirements. See *Oracle Machine Learning for SQL Concepts* for additional information about class weights.
Classification	`CLAS_WEIGHTS_BALANCED`	`ON` `OFF`	Indicates balancing of target distribution in the model. Relevant for rare targets; can improve average accuracy (average of per-class accuracy instead of overall accuracy which favors the dominant class). Particularly useful in data sets with imbalanced classes, ensuring rare events are adequately captured in the model. The default value is `OFF`.

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description

Classification	CLAS_MAX_SUP_BINS	For Decision Tree:
		Specify an integer between 2 and 2147483647, inclusive represented as a character string.
		For Random Forest:
		Specify an integer between 2 and 254, inclusive represented as a character string.
			Specifies the maximum number of bins for each attribute. Manages the granularity of data binning, affecting model complexity and potentially influencing model accuracy and computation time.
			The default value is 32.
			Expression:
			For Decision Tree:
			`2 <= a number <=2147483647`
			For Random Forest:
			`2 <= a number <=254`
			See, DBMS_DATA_MINING — Automatic Data Preparation

Clustering	CLUS_NUM_CLUSTERS	An integer greater than or equal to 1 expressed as a character string	The maximum number of leaf clusters generated by a clustering algorithm. The algorithm may return fewer clusters, depending on the data. Expression: `TO_CHAR(9)` Enhanced *k*-Means usually produces the exact number of clusters specified by CLUS_NUM_CLUSTERS, unless there are fewer distinct data points. When Expectation maximization (EM) is used for clustering, it may return fewer clusters than the number specified by CLUS_NUM_CLUSTERS depending on the data. The number of clusters returned by EM cannot be greater than the number of components, which is governed by algorithm-specific settings. (See *Expectation Maximization Settings for Learning* table) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, the number of clusters equals the number of components. The setting can be used only for EM Clustering algorithm. For EM Clustering algorithm, the default value of CLUS_NUM_CLUSTERS is system-determined. For *k*-Means and O-Cluster, the default is 10.

Table 62-7 (Cont.) Machine Learning Function Settings

Machine Learning Function	Setting Name	Setting Value	Description
Feature extraction	`FEAT_NUM_FEATURES`	An integer greater than or equal to 1 expressed as a character string	The number of features to be extracted by a feature extraction model. Expression: `TO_CHAR(8)` The default is estimated from the data by the algorithm. If the matrix rank is smaller than this number, fewer features will be returned. For CUR Matrix Decomposition, the `FEAT_NUM_FEATURES` value is the same as the `CURS_SVD_RANK` value.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about machine learning functions

DBMS_DATA_MINING — Global Settings

The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

Table 62-8 Global Settings

Setting Name	Setting Value	Description
`ODMS_BOXCOX`	`ODMS_BOXCOX_ENABLE` `ODMS_BOXCOX_DISABLE`	This setting enables the Box-Cox variance-stabilization transformation. It is useful when the variance increases as the target value increases. It reduces variance and transforms a multiplicative relationship with the target, with a simpler additive relationship. This setting is applicable only to the Exponential Smoothing algorithm. When a value for `EXSM_MODEL` setting is not specified, the default value is `ODMS_BOXCOX_ENABLE` and when a value for the `EXSM_MODEL` setting is provided, the default value is `ODMS_BOXCOX_DISABLE`.
`ODMS_EXPLOSION_MIN_SUPP`	A positive integer	It is the minimum required support for categorical values that must be included in the explosion mapping. It removes categorical values with insufficient row instances to have a statistically significant effect on the model, however, they could potentially degrade performance. The default is system determined depending on the number of rows in the dataset. A value of `1` results into mapping all categorical values.

Table 62-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_ITEM_ID_COLUMN_NAME	*column_name*	(Association rules only) Name of a column that contains the items in a transaction. When this setting is specified, the algorithm expects the data to be presented in a native transactional format, consisting of two columns: • Case ID, either categorical or numeric • Item ID, either categorical or numeric **Note:** Oracle Machine Learning does not support BOOLEAN values for this setting. A typical example of transactional data is market basket data, wherein a case represents a basket that may contain many items. Each item is stored in a separate row, and many rows may be needed to represent a case. The case ID values do not uniquely identify each row. Transactional data is also called multi-record case data. Association rules function is normally used with transactional data, but it can also be applied to single-record case data (similar to other algorithms). For more information about single-record and multi-record case data, see *Oracle SQL Developer Data Modeler User's Guide*.

Table 62-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_ITEM_VALUE_COLUMN_NAME	*column_name*	(Association rules only) Name of a column that contains a value associated with each item in a transaction. This setting is only used when a value has been specified for ODMS_ITEM_ID_COLUMN_NAME indicating that the data is presented in native transactional format. If ASSO_AGGREGATES is used, then the build data must include the following three columns and the columns specified in the AGGREGATES setting. • Case ID, either categorical or numeric • Item ID, either categorical or numeric, specified by ODMS_ITEM_ID_COLUMN_NAME • Item value, either categorical or numeric, specified by ODMS_ITEM_VALUE_COLUMN_NAME **Note:** Oracle Machine Learning does not support BOOLEAN values for this setting. If ASSO_AGGREGATES, Case ID, and Item ID column are present, then the Item Value column may or may not appear. The Item Value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples). For details on ASSO_AGGREGATES, see DBMS_DATA_MINING - Mining Function Settings.
ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_MEAN_MODE ODMS_MISSING_VALUE_DELETE_ROW ODMS_MISSING_VALUE_AUTO	Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO. ODMS_MISSING_VALUE_MEAN_MODE replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time where appropriate. ODMS_MISSING_VALUE_AUTO performs different strategies for different algorithms. When ODMS_MISSING_VALUE_TREATMENT is set to ODMS_MISSING_VALUE_DELETE_ROW, the rows in the training data that contain missing values are deleted. However, if you want to replicate this missing value treatment in the scoring data, then you must perform the transformation explicitly. The value ODMS_MISSING_VALUE_DELETE_ROW applies to all algorithms.

Table 62-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_ROW_WEIGHT_COLUMN_NAME	*column_name*	(GLM only) Name of a column in the training data that contains a weighting factor for the rows. The column data type must be numeric. Oracle Machine Learning does not support BOOLEAN values for this setting. Row weights can be used as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. Row weights can also be used to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.
ODMS_TEXT_POLICY_NAME	The name of an Oracle Text POLICY created using CTX_DDL.CREATE_POLICY.	Affects how individual tokens are extracted from unstructured text. For details about CTX_DDL.CREATE_POLICY, see *Oracle Text Reference*.
ODMS_TEXT_MAX_FEATURES	1 <= *value*	The maximum number of distinct features, across all text attributes, to use from a document set passed to CREATE_MODEL. The default is 3000. ESA has the default value of 300000.
ODMS_TEXT_MIN_DOCUMENTS	Non-negative value	This is a text processing setting the controls how in how many documents a token needs to appear to be used as a feature. The default is 1. ESA has a default of 3.
ODMS_PARTITION_COLUMNS	Comma separated list of machine learning attributes	This setting indicates a request to build a partitioned model. The setting value is a comma-separated list of the machine learning attributes used to determine the in-list partition key values. Oracle Machine Learning supports numeric and categorical values including BOOLEAN for this setting. These machine learning attributes are taken from the input columns unless an XFORM_LIST parameter is passed to CREATE_MODEL or CREATE_MODEL2. If the XFORM_LIST parameter is passed to during model building, then the machine learning attributes are taken from the attributes produced by these transformations.
ODMS_MAX_PARTITIONS	1< value <= 1000000	This setting indicates the maximum number of partitions allowed for the model. The default is 1000.
ODMS_SAMPLING	ODMS_SAMPLING_ENABLE ODMS_SAMPLING_DISABLE	This setting allows the user to request a sampling of the build data. The default is ODMS_SAMPLING_DISABLE.
ODMS_SAMPLE_SIZE	0 < Value	This setting determines how many rows will be sampled (approximately). It can be set only if ODMS_SAMPLING is enabled. The default value is the system determined.

Table 62-8 (Cont.) Global Settings

Setting Name	Setting Value	Description
ODMS_PARTITION_BUILD_TYPE	ODMS_PARTITION_BUILD_INTRA ODMS_PARTITION_BUILD_INTER ODMS_PARTITION_BUILD_HYBRID	This setting controls the parallel build of partitioned models. ODMS_PARTITION_BUILD_INTRA — Each partition is built in parallel using all replicas. ODMS_PARTITION_BUILD_INTER — Each partition is built entirely in a single slave, but multiple partitions may be built at the same time since multiple replicas are active. ODMS_PARTITION_BUILD_HYBRID — It is a combination of the other two types and is recommended for most situations to adapt to dynamic environments. The default mode is ODMS_PARTITION_BUILD_HYBRID
ODMS_TABLESPACE_NAME	*tablespace_name*	This setting controls the storage specifications. If you explicitly sets this to the name of a tablespace (for which you have sufficient quota), then the specified tablespace storage creates the resulting model content. If you do not provide this setting, then the default tablespace of the user creates the resulting model content.
ODMS_RANDOM_SEED	The value must be a non-negative integer	The hash function with a random number seed generates a random number with uniform distribution. Users can control the random number seed by this setting. The default is 0. This setting is used by Random Forest, Neural Network, and CUR Matrix Decomposition.
ODMS_DETAILS	• ODMS_ENABLE • ODMS_DISABLE	This setting reduces the space that is used while creating a model, especially a partitioned model. The default value is ODMS_ENABLE. When the setting is ODMS_ENABLE, it creates model tables and views when the model is created. You can query the model with SQL. When the setting is ODMS_DISABLE, model views are not created and tables relevant to model details are not created either. The reduction in space depends on the model. Reduction on the order of 10x can be achieved.

See Also:

Oracle Machine Learning for SQL Concepts for information about GLM

Oracle Machine Learning for SQL Concepts for information about association rules

Oracle Machine Learning for SQL User's Guide for information about machine learning unstructured text

DBMS_DATA_MINING — Algorithm Specific Model Settings

Oracle Machine Learning for SQL uses algorithm specific settings to define the characteristics of a model.

All settings have default values. If you want to override one or more of the settings for a model, then you must specify those settings.

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the CREATE_MODEL Procedure. You can also provide these settings through the CREATE_MODEL2 Procedure.

The settings used by a model can be found by querying the data dictionary view `ALL_MINING_MODEL_SETTINGS`. This view displays the model settings used by the machine learning models to which you have access. All of the default and user-specified setting values are included in the view.

> **See Also:**
>
> - `ALL_MINING_MODEL_SETTINGS` in *Oracle Database Reference*
> - *Oracle Machine Learning for SQL User's Guide* for information about specifying model settings

DBMS_DATA_MINING — Algorithm Settings: ALGO_EXTENSIBLE_LANG

The settings listed in the following table configure the behavior of the machine learning model with an extensible algorithm. The model is built in the R language.

The `RALG_*_FUNCTION` specifies the R script that is used to build, score, and view an R model and must be registered in the Oracle Machine Learning for R script repository. The R scripts are registered through Oracle Machine Learning for R with special privileges. When `ALGO_EXTENSIBLE_LANG` is set to R in the `MINING_MODEL_SETTING` table, the machine learning model is built in the R language. After the R model is built, the names of the R scripts are recorded in the `MINING_MODEL_SETTING` table in the `SYS` schema. The scripts must exist in the script repository for the R model to function. The amount of R memory used to build, score, and view the R model through these R scripts can be controlled by Oracle Machine Learning for R.

All algorithm-independent `DBMS_DATA_MINING` subprograms can operate on an R model for machine learning functions such as association, attribute importance, classification, clustering, feature extraction, and regression.

The supported `DBMS_DATA_MINING` subprograms include, but are not limited, to the following:

- ADD_COST_MATRIX Procedure
- COMPUTE_CONFUSION_MATRIX Procedure
- COMPUTE_LIFT Procedure
- COMPUTE_ROC Procedure
- CREATE_MODEL Procedure
- DROP_MODEL Procedure

- EXPORT_MODEL Procedure
- GET_MODEL_COST_MATRIX Function
- IMPORT_MODEL Procedure
- REMOVE_COST_MATRIX Procedure
- RENAME_MODEL Procedure

Table 62-9 ALGO_EXTENSIBLE_LANG Settings

Setting Name	Setting Value	Description
`RALG_BUILD_FUNCTION`	`R_BUILD_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm machine learning model build function. The R script defines an R function for the first input argument for training data and returns an R model object. For clustering and feature extraction machine learning function model build, the R attributes `dm$nclus` and `dm$nfeat` must be set on the R model to indicate the number of clusters and features respectively. The `RALG_BUILD_FUNCTION` must be set along with `ALGO_EXTENSIBLE_LANG` in the `model_setting_table`.
`RALG_BUILD_PARAMETER`	`SELECT` *value* `param_name`, ...`FROM DUAL`	Specifies a list of numeric and string scalar for optional input parameters of the model build function.
`RALG_SCORE_FUNCTION`	`R_SCORE_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script to score data. The script returns a `data.frame` containing the corresponding prediction results. The setting is used to score data for machine learning functions such as regression, classification, clustering, and feature extraction. This setting does not apply to the association and the attribute importance functions.
`RALG_WEIGHT_FUNCTION`	`R_WEIGHT_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm that computes the weight (contribution) for each attribute in scoring. The script returns a `data.frame` containing the contributing weight for each attribute in a row. This function setting is needed for the `PREDICTION_DETAILS` SQL function.
`RALG_DETAILS_FUNCTION`	`R_DETAILS_FUNCTION_SCRIPT_NAME`	Specifies the name of an existing registered R script for the R algorithm that produces the model information. This setting is required to generate a model view.
`RALG_DETAILS_FORMAT`	`SELECT` *type_value* *column_name*, ... `FROM DUAL`	Specifies the `SELECT` query for the list of numeric and string scalars for the output column type and the column name of the generated model view. This setting is required to generate a model view.

See Also:

Oracle Machine Learning for SQL User's Guide

DBMS_DATA_MINING — Algorithm Settings: CUR Matrix Decomposition

The following settings affects the behavior of the CUR Matrix Decomposition algorithm.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.CURS_ROW_IMP_DISABLE`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'CURS_ROW_IMP_DISABLE'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-10 CUR Matrix Decomposition Settings

Setting Name	Constant Value	String Value Equivalent	Description
`CURS_APPROX_ATTR_NUM`	The value must be a positive integer	The value must be a positive integer	Defines the approximate number of attributes to be selected. The default value is the number of attributes.
`CURS_ROW_IMPORTANCE`	`CURS_ROW_IMP_ENABLE`	`CURS_ROW_IMP_ENABLE`	Defines the flag indicating whether or not to perform row selection. Enables row selection. The default value is `CURS_ROW_IMP_DISABLE`.
	`CURS_ROW_IMP_DISABLE`	`CURS_ROW_IMP_DISABLE`	Disables row selection.
`CURS_APPROX_ROW_NUM`	The value must be a positive integer	The value must be a positive integer	Defines the approximate number of rows to be selected. This parameter is only used when users decide to perform row selection (`CURS_ROW_IMP_ENABLE`). The default value is the total number of rows.
`CURS_SVD_RANK`	The value must be a positive integer	The value must be a positive integer	Defines the rank parameter used in the column/row leverage score calculation. If users do not provide an input value, the value is determined by the system.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts

DBMS_DATA_MINING — Algorithm Settings: Decision Tree

These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual decision tree.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.TREE_IMPURITY_ENTROPY`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'TREE_IMPURITY_ENTROPY'`.

Note:

The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-11 Decision Tree Settings

Setting Name	Constant Value	String Value Equivalent	Description
`TREE_IMPURITY_METRIC`	`TREE_IMPURITY_ENTROPY`	`TREE_IMPURITY_ENTROPY`	Tree impurity metric for Decision Tree. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split values are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is by a metric. By default, the algorithm uses `TREE_IMPURITY_GINI`.
	`TREE_IMPURITY_GINI`	`TREE_IMPURITY_GINI`	Decision trees can use either Gini (`TREE_IMPURITY_GINI`) or entropy (`TREE_IMPURITY_ENTROPY`) as the purity metric.

`TREE_TERM_MAX_DEPTH`	For Decision Tree:
	`2<=` *`a number`* `<=20`
	For Random Forest:
	`2<=` *`a number`* `<=100`
		For Decision Tree:
		`2<=` *`a number`* `<=20`
		For Random Forest:
		`2<=` *`a number`* `<=100`
			Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node).
			For Decision Tree, the default is `7`.
			For Random Forest, the default is `16`.

`TREE_TERM_MINPCT_NODE`	`0<=` *`a number`*`<=10`	`0<=` *`a number`*`<=10`	The minimum number of training rows in a node expressed as a percentage of the rows in the training data. Default is `0.05`, indicating 0.05%.

Table 62-11 (Cont.) Decision Tree Settings

Setting Name	Constant Value	String Value Equivalent	Description
TREE_TERM_MINPCT_SPLIT	0 < *a number* <=20	0 < *a number* <=20	The minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. Default is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	*a number*>=0	*a number*>=0	The minimum number of rows in a node. Default is 10.
TREE_TERM_MINREC_SPLIT	a number > 1	a number > 1	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. Default is 20.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about Decision Tree

DBMS_DATA_MINING — Algorithm Settings: Expectation Maximization

These algorithm settings configure the behavior of the Expectation Maximization algorithm.

See Also:

Oracle Data Mining Concepts for information about Expectation Maximization

Table 62-12 Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Constant Value	String Value Equivalent	Description
EMCS_ATTRIBUTE_FILTER	EMCS_ATTR_FILTER_ENABLE	EMCS_ATTR_FILTER_ENABLE	Whether or not to include uncorrelated attributes in the model. When EMCS_ATTRIBUTE_FILTER is enabled, uncorrelated attributes are not included. **Note:** This setting applies only to attributes that are not nested. For Clustering, the default is system-determined. For anomaly detection, the default is EMCS_ATTR_FILTER_DISABLE.
	EMCS_ATTR_FILTER_DISABLE	EMCS_ATTR_FILTER_DISABLE	Includes uncorrelated attributes in the model.
EMCS_MAX_NUM_ATTR_2D	An integer greater than or equal to 1, represented as a character string.	An integer greater than or equal to 1, represented as a character string.	Maximum number of correlated attributes to include in the model. Note: This setting applies only to attributes that are not nested (2D). Default is 50. Expression: TO_CHAR(40)
EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_BERNOULLI	EMCS_NUM_DISTR_BERNOULLI	The distribution for modeling numeric attributes. Applies to the input table or view as a whole and does not allow per-attribute specifications. The options include Bernoulli, Gaussian, or system-determined distribution. When Bernoulli or Gaussian distribution is chosen, all numeric attributes are modeled using the same type of distribution. Default is EMCS_NUM_DISTR_SYSTEM.
	EMCS_NUM_DISTR_GAUSSIAN	EMCS_NUM_DISTR_GAUSSIAN	Models all numeric attributes using Gaussian distribution.
	EMCS_NUM_DISTR_SYSTEM	EMCS_NUM_DISTR_SYSTEM	When the distribution is system-determined, individual attributes may use different distributions (either Bernoulli or Gaussian), depending on the data.
EMCS_NUM_EQUIWIDTH_BINS	An integer between 1 to 255, inclusive, represented as a character string.	An integer between 1 to 255, inclusive, represented as a character string.	Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. Default is 11. Expression: TO_CHAR(20)

Table 62-12 (Cont.) Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Constant Value	String Value Equivalent	Description
EMCS_NUM_PROJECTIONS	An integer greater than or equal to 1, represented as a character string.	An integer greater than or equal to 1, represented as a character string.	Specifies the number of projections that will be used for each nested column. If a column has fewer distinct attributes than the specified number of projections, the data will not be projected. The setting applies to all nested columns. Default is 50. Expression: TO_CHAR(40)
EMCS_NUM_QUANTILE_BINS	An integer between 1 to 255, inclusive, represented as a character string.	An integer between 1 to 255, inclusive, represented as a character string.	Specifies the number of quantile bins that will be used for modeling numeric columns with multivalued Bernoulli distributions. Default is system-determined. Expression: TO_CHAR(20)
EMCS_NUM_TOPN_BINS	An integer between 1 to 255, inclusive, represented as a character string.	An integer between 1 to 255, inclusive, represented as a character string.	Specifies the number of top-N bins that will be used for modeling categorical columns with multivalued Bernoulli distributions. Default is system-determined. Expression: TO_CHAR(10)

Table 62-13 Expectation Maximization Settings for Learning

Setting Name	Constant Value	String Value Equivalent	Description
EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_HELDASIDE	EMCS_CONV_CRIT_HELDASIDE	The convergence criterion for EM. The convergence criterion may be based on a held-aside data set, or it may be Bayesian Information Criterion. EMCS_CONV_CRIT_HELDASIDE: Uses a held-aside data set for convergence criterion. Default is system determined.
	EMCS_CONV_CRIT_BIC	EMCS_CONV_CRIT_BIC	Uses the Bayesian Information Criterion (BIC) for convergence.
EMCS_LOGLIKE_IMPROVEMENT	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	When the convergence criterion is based on a held-aside data set (EMCS_CONVERGENCE_CRITERION = EMCS_CONV_CRIT_HELDASIDE), this setting specifies the percentage improvement in the value of the log likelihood function that is required for adding a new component to the model. Default value is 0.001. Expression: TO_CHAR(0.003)

Table 62-13 (Cont.) Expectation Maximization Settings for Learning

Setting Name	Constant Value	String Value Equivalent	Description
`EMCS_NUM_COMPONENTS`	An integer greater than or equal to 1, represented as a character string	An integer greater than or equal to 1, represented as a character string	Maximum number of components in the model. If model search is enabled, the algorithm automatically determines the number of components based on improvements in the likelihood function or based on regularization, up to the specified maximum. For EM Clustering, the number of components must be greater than or equal to the number of clusters. Default is 20 for both EM Clustering and EM Anomaly. Expression: `TO_CHAR(20)`
`EMCS_NUM_ITERATIONS`	An integer greater than or equal to 1, represented as a character string	An integer greater than or equal to 1, represented as a character string	Specifies the maximum number of iterations in the EM algorithm. Default is `100`. Expression: `TO_CHAR(50)`
`EMCS_MODEL_SEARCH`	`EMCS_MODEL_SEARCH_ENABLE`	`EMCS_MODEL_SEARCH_ENABLE`	This setting enables model search in EM where different model sizes are explored and a best size is selected. The default is `EMCS_MODEL_SEARCH_DISABLE`.
	`EMCS_MODEL_SEARCH_DISABLE` (default).	`EMCS_MODEL_SEARCH_DISABLE` (default).	The model search in EM is disabled.
`EMCS_REMOVE_COMPONENTS`	`EMCS_REMOVE_COMPS_ENABLE` (default)	`EMCS_REMOVE_COMPS_ENABLE` (default)	This setting allows the EM algorithm to remove a small component from the solution. The default is `EMCS_REMOVE_COMPS_ENABLE`.
	`EMCS_REMOVE_COMPS_DISABLE`	`EMCS_REMOVE_COMPS_DISABLE`	Prevents the EM algorithm from removing small components.
`EMCS_RANDOM_SEED`	Non-negative integer	Non-negative integer	This setting controls the seed of the random generator used in EM. The default is `0`.

Table 62-14 Expectation Maximization Settings for Component Clustering

Setting Name	Constant Value	String Value Equivalent	Description
`EMCS_CLUSTER_COMPONENTS`	`EMCS_CLUSTER_COMP_ENABLE`	`EMCS_CLUSTER_COMP_ENABLE`	Enables or disables the grouping of EM components into high-level clusters. When disabled, the components themselves are treated as clusters. The setting can be used only for EM Clustering. When component clustering is enabled, model scoring through the SQL `CLUSTER` function will produce assignments to the higher level clusters. Default is `EMCS_CLUSTER_COMP_ENABLE`.

Table 62-14 (Cont.) Expectation Maximization Settings for Component Clustering

Setting Name	Constant Value	String Value Equivalent	Description
	EMCS_CLUSTER_COMP_DISABLE	EMCS_CLUSTER_COMP_DISABLE	When clustering is disabled, the CLUSTER function will produce assignments to the original components.
EMCS_CLUSTER_THRESH	Specify an integer greater than or equal to 1, represented as a character string	Specify an integer greater than or equal to 1, represented as a character string	Dissimilarity threshold that controls the clustering of EM components. When the dissimilarity measure is less than the threshold, the components are combined into a single cluster. The setting can be used only for EM Clustering. A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out. Default is 2. Expression: TO_CHAR(3)
EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE	EMCS_LINKAGE_SINGLE	Allows the specification of a linkage function for the agglomerative clustering step. EMCS_LINKAGE_SINGLE uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes. Default is EMCS_LINKAGE_SINGLE.
	EMCS_LINKAGE_AVERAGE	EMCS_LINKAGE_AVERAGE	EMCS_LINKAGE_AVERAGE uses the average distance within the branch. There is less chaining effect and the clusters are more compact.
	EMCS_LINKAGE_COMPLETE	EMCS_LINKAGE_COMPLETE	EMCS_LINKAGE_COMPLETE uses the maximum distance within the branch. The clusters are smaller and require strong component overlap.

Table 62-15 Expectation Maximization Settings for Cluster Statistics

Setting Name	Constant Value	Description
EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE EMCS_CLUS_STATS_DISABLE	Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced, and GET_MODEL_DETAILS_EM only returns taxonomy (hierarchy) and cluster counts. The setting can be used only for EM Clustering. Default is EMCS_CLUS_STATS_ENABLE.
EMCS_MIN_PCT_ATTR_SUPPORT	A floating point number between 0 and 1 expressed as a character string	Minimum support required for including an attribute in the cluster rule. The support is the percentage of the data rows assigned to a cluster that must have non-null values for the attribute. The setting can be used only for EM Clustering. Default is 0.1. Expression: TO_CHAR(0.9)

Table 62-16 Expectation Maximization Settings for Anomaly Detection

Setting Name	Constant Value	String Value Equivalent	Description
`EMCS_OUTLIER_RATE`	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	The desired rate of outliers in the training data. The setting can be used only for EM Anomaly. Default is 0.05. Expression: `TO_CHAR(0.07)`

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

DBMS_DATA_MINING — Algorithm Settings: Explicit Semantic Analysis

Explicit Semantic Analysis (ESA) is a useful technique for extracting meaningful and interpretable features.

The settings listed in the following table configure the ESA values.

Table 62-17 Explicit Semantic Analysis Settings

Setting Name	Setting Value	String Value Equivalent	Description
`ESAS_EMBEDDINGS`	`ESAS_EMBEDDINGS_ENABLE`	`ESAS_EMBEDDINGS_ENABLE`	This setting applies to feature extraction models. The default value is `ESAS_EMBEDDINGS_DISABLE`. When you set `ESAS_EMBEDDINGS_ENABLE`: • ESA generates embeddings during scoring • The FEATURE_ID of the generated embeddings is of the data type NUMBER • The `CASE_ID_COLUMN_NAME` argument of the `DBMS_DATA_MINING.CREATE_MODEL` and `DBMS_DATA_MINING.CREATE_MODEL2` function is optional.
	`ESAS_EMBEDDINGS_DISABLE`	`ESAS_EMBEDDINGS_DISABLE`	Disables the use of embeddings for ESA. This setting is useful when embeddings are not required or desired for the analysis

Table 62-17 (Cont.) Explicit Semantic Analysis Settings

Setting Name	Setting Value	String Value Equivalent	Description
ESAS_EMBEDDING_SIZE	A positive integer less than or equal to 4096	A positive integer less than or equal to 4096	This setting applies to feature extraction models. This setting specifies the size of the vectors representing embeddings. You can set this parameter only if you have enabled ESAS_EMBEDDINGS. The default size is 1024. If this value is less than the number of distinct features in the training set, then the actual number of explicit features is used as the size of embedding vectors instead.
ESAS_MIN_ITEMS	Text input 100 Non-text input is 0	Text input 100 Non-text input is 0	This setting determines the minimum number of non-zero entries that need to be present in an input row. The default is 100 for text input and 0 for non-text input.
ESAS_TOPN_FEATURES	A positive integer	A positive integer	This setting controls the maximum number of features per attribute. The default is 1000.
ESAS_VALUE_THRESHOLD	Non-negative number	Non-negative number	This setting thresholds a small value for attribute weights in the transformed build data. The default is 1e-8.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about ESA.

DBMS_DATA_MINING — Algorithm Settings: Exponential Smoothing

These settings configure the behavior of the Exponential Smoothing (ESM) algorithm.

The settings listed in the following table specify the setting names and possible values for Exponential Smoothing. You can specify the Setting Value using the prefix DBMS_DATA_MINING. For example, DBMS_DATA_MINING.EXSM_SIMPLE. Alternatively, you can specify the Setting Value without the DBMS_DATA_MINING prefix, in single quotes. For example, 'EXSM_SIMPLE'.

For Global settings, see DBMS_DATA_MINING — Global Settings.

Table 62-18 Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_MODEL	EXSM_SIMPLE EXSM_SIMPLE_MULT_ERR EXSM_HOLT EXSM_HOLT_DAMPED EXSM_MULT_TREND EXSM_MULT_TREND_DAMPED EXSM_SEASON_ADD EXSM_SEASON_MUL EXSM_WINTERS EXSM_WINTERS_DAMPED EXSM_ADDWINTERS EXSM_ADDWINTERS_DAMPED EXSM_WINTERS_MUL_TREND EXSM_WINTERS_MUL_TREND_DMP	This setting specifies the model. EXSM_SIMPLE: Forecasts data as a weighted moving average, with the influence of past observations declining exponentially with the length of time since the observation occurred. Errors in estimation are assumed to be normally distributed, with constant mean and variance. It is appropriate for data with no clear trend or seasonal pattern. EXSM_SIMPLE_MULT_ERR: Forecasts data as a weighted moving average, with the influence of past observations declining exponentially with the length of time since the observation occurred. Errors in estimation are assumed to be proportional to the level of the prior estimate. EXSM_HOLT: Applies Holt's linear exponential smoothing method, designed to forecast data with an underlying linear trend. EXSM_HOLT_DAMPED: Applies Holt's linear exponential smoothing with a damping factor to progressively reduce the strength of the trend over time. EXSM_MULT_TREND: Applies an exponential smoothing framework with a multiplicative trend component, effectively capturing data where trends are not linear but grow or decay over time. EXSM_MULT_TREND_DAMPED: Applies an exponential smoothing algorithm with a multiplicative trend that diminishes over time, providing a conservative approach to trend estimation. EXSM_SEASON_ADD: Applies an exponential smoothing with an additive seasonal component, isolating and accounting for seasonal variations without incorporating a trend. EXSM_SEASON_MUL: Executes exponential smoothing with a multiplicative seasonal component, capturing seasonal effects that increase or decrease in proportion to the level of the series. EXSM_WINTERS: Applies the Holt-Winters method with additive trends and multiplicative seasonality, offering a robust model for data with both linear trend and proportional seasonal variation. EXSM_WINTERS_DAMPED: Applies the Holt-Winters method with a damped trend and multiplicative seasonality, moderating the linear trend over time while still capturing proportional seasonal changes.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
		EXSM_ADDWINTERS: Applies the Holt-Winters additive model to simultaneously smooth data with linear trends and additive seasonal effects. EXSM_ADDWINTERS_DAMPED: Applies the Holt-Winters additive approach with a damping mechanism, reducing the impact of the trend and seasonal components over time. EXSM_WINTERS_MULT_TREND: Applies the Holt-Winters model with both trend and seasonality components being multiplicative, suited for series where the seasonal variations and trends are both increasing or decreasing proportional to level. EXSM_WINTERS_MUL_TREND_DMP: Applies the Holt-Winters model with a damped multiplicative trend, effectively moderating the exponential increase or decrease of both trend and seasonal components over time. The default value is EXSM_SIMPLE.
EXSM_SEASONALITY	positive integer > 1	This setting specifies a positive integer value as the length of seasonal cycle. The value it takes must be larger than 1. For example, setting value 4 means that every group of four observations forms a seasonal cycle. This setting is only applicable and must be provided for models with seasonality, otherwise the model throws an error. When EXSM_INTERVAL is not set, this setting applies to the original input time series. When EXSM_INTERVAL is set, this setting applies to the accumulated time series.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_INTERVAL	EXSM_INTERVAL_YEAR EXSM_INTERVAL_QTR EXSM_INTERVAL_MONTH EXSM_INTERVAL_WEEK EXSM_INTERVAL_DAY EXSM_INTERVAL_HOUR EXSM_INTERVAL_MINUTE EXSM_INTERVAL_SECOND	This setting only applies and must be provided when the time column (case_id column) has datetime type. It specifies the spacing interval of the accumulated equally spaced time series. The model throws an error if the time column of input table is of datetime type and setting EXSM_INTERVAL is not provided. The model throws an error if the time column of input table is of oracle number type and setting EXSM_INTERVAL is provided. EXSM_INTERVAL_YEAR: This option sets the spacing interval of the accumulated time series to one year. When selected, the data is aggregated or summarized on a yearly basis. EXSM_INTERVAL_QTR: This option sets the spacing interval to a quarter, aggregating the data for every three months. EXSM_INTERVAL_MONTH: This option adjusts the spacing interval to one month. The accumulated time series represent aggregated or summarized data for each month. EXSM_INTERVAL_WEEK: With this option data is aggregated or summarized on a weekly basis, setting the spacing interval to one week. EXSM_INTERVAL_DAY: This option adjusts the spacing interval to one day. It's suitable for scenarios where daily aggregated insights are required. EXSM_INTERVAL_HOUR: For more granular insights, this option sets the spacing interval to one hour. It's especially useful when analyzing data that changes significantly within a day. EXSM_INTERVAL_MINUTE: With this option the spacing is set to one minute. This provides a very detailed view of data, suitable for applications like high-frequency trading or real-time monitoring systems. EXSM_INTERVAL_SECOND: For most granular details, this options sets the spacing interval to one second. It's tailored for scenarios requiring real-time or near-real-time analysis.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_INITVL_OPTIMIZE	EXSM_INITVL_OPTIMIZE_ENABLE EXSM_INITVL_OPTIMIZE_DISABLE	The setting EXSM_INITVL_OPTIMIZE determines whether initial values are optimized during model build. The default value is EXSM_INITVL_OPTIMIZE_ENABLE. **Note:** EXSM_INITVL_OPTIMIZE can only be set to EXSM_INITVL_OPTIMIZE_DISABLE if the user has set EXSM_MODEL to EXSM_HW or EXSM_HW_ADDSEA. If EXSM_MODEL is set to another model type or is not specified, error 40213 (conflicting settings) is thrown and the model is not built.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
EXSM_ACCUMULATE	EXSM_ACCU_TOTAL EXSM_ACCU_STD EXSM_ACCU_MAX EXSM_ACCU_MIN EXSM_ACCU_AVG EXSM_ACCU_MEDIAN EXSM_ACCU_COUNT	This setting only applies and must be provided when the time column has datetime type. It specifies how to generate the value of the accumulated time series from the input time series. EXSM_ACCU_TOTAL: This option calculates the total sum of the time series values within a specified interval. When selected, it will aggregate the data by summing up all the individual values in the datetime range. EXSM_ACCU_STD: This option computes the standard deviation of the time series values within a specified interval. It helps you understand the amount of variation or dispersion in your data. EXSM_ACCU_MAX: By selecting this option, the maximum value of the time series within a specified interval will be determined. It helps in identifying the peak value in the given range. EXSM_ACCU_MIN: This option focuses on determining the minimum value of the time series within a specified interval. It is useful for identifying the lowest value in the time series for the given datetime range. EXSM_ACCU_AVG: This specifies the average value of your time series within a specified interval. It calculates the mean value of all data points in the specified range. EXSM_ACCU_MEDIAN: This option provides the median of the time series values within the given interval. The median gives a central value, which can be especially useful if your data contains outliers. EXSM_ACCU_COUNT: This option counts the number of time series values within the specified interval. It is helpful if you want to know how many data points are present in a certain datetime range. The default value is EXSM_ACCU_TOTAL.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description

`EXSM_SETMISSING`	Specify an option:
	`EXSM_MISS_MIN`
	`EXSM_MISS_MAX`
	`EXSM_MISS_AVG`
	`EXSM_MISS_MEDIAN`
	`EXSM_MISS_LAST`
	`EXSM_MISS_FIRST`
	`EXSM_MISS_PREV`
	`EXSM_MISS_NEXT`
	`EXSM_MISS_AUTO`
		This setting specifies how to handle missing values, which may come from input data and/or the accumulation process of time series. You can specify either a number or an option. If a number is specified, all the missing values are set to that number.
		`EXSM_MISS_MIN`: Replaces missing value with minimum of the accumulated time series.
		`EXSM_MISS_MAX`: Replaces missing value with maximum of the accumulated time series.
		`EXSM_MISS_AVG`: Replaces missing value with average of the accumulated time series.
		`EXSM_MISS_MEDIAN`: Replaces missing value with median of the accumulated time series.
		`EXSM_MISS_LAST`: Replaces missing value with last non-missing value of the accumulated time series.
		`EXSM_MISS_FIRST`: Replaces missing value with first non-missing value of the accumulated time series.
		`EXSM_MISS_PREV`: Replaces missing value with the previous non-missing value of the accumulated time series.
		`EXSM_MISS_NEXT`: Replaces missing value with the next non-missing value of the accumulated time series.
		`EXSM_MISS_AUTO`: EXSM model treats the input data as an irregular (non-uniformly spaced) time series.
		If this setting is not provided, `EXSM_MISS_AUTO` is the default value. In such a case, the model treats the input time series as irregular time series, viewing missing values as gaps.

`EXSM_PREDICTION_STEP`	It must be set to a number between 1-30.	This setting specifies how many steps ahead the predictions are to be made. If it is not set, the default value is `1`: the model gives one-step-ahead prediction. A value greater than `30` results in an error.
`EXSM_CONFIDENCE_LEVEL`	It must be a number between 0 and 1, exclusive.	This setting specifies the desired confidence level for prediction. The lower and upper bounds of the specified confidence interval is reported. If this setting is not specified, the default confidence level is `95%`.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description

`EXSM_OPT_CRITERION`	`EXSM_OPT_CRIT_LIK`
	`EXSM_OPT_CRIT_MSE`
	`EXSM_OPT_CRIT_AMSE`
	`EXSM_OPT_CRIT_SIG`
	`EXSM_OPT_CRIT_MAE`
		This setting specifies the desired optimization criterion. The optimization criterion is useful as a diagnostic for comparing models' fit to the same data.
		`EXSM_OPT_CRIT_LIK`: This represents the negative double of the logarithm of the likelihood associated with a given model.
		`EXSM_OPT_CRIT_MSE`: This provides the mean squared error pertaining to the model.
		`EXSM_OPT_CRIT_AMSE`: This denotes the average of the mean squared error over a time window as specified by the user.
		`EXSM_OPT_CRIT_SIG`: This metric captures the standard deviation of the residuals of the model.
		`EXSM_OPT_CRIT_MAE`: This metric conveys the average absolute error associated with the model. It measures the size of the error.
		The default value is `EXSM_OPT_CRIT_LIK`.

`EXSM_NMSE`	positive integer	This setting specifies the length of the window used in computing the error metric average mean square error (AMSE).
`EXSM_SERIES_LIST`	Comma delimited list of time series columns	This setting allows you to forecast up to twenty predictor series in addition to the target series. The column names in `EXSM_SERIES_LIST` are enclosed in single quotes. **Note:** The list is enclosed in single quotes, not the individual column names. For example: `INSERT INTO <settings_table_name VALUES(dbms_data_mining.exsm_series_list, ‘<column1>,<column2>,<column3>,<column4>’);` The prefix `DM$` must be added to the build and scoring data sets. The column names must be less than 125 characters long. See Model Detail Views for Exponential Smoothing.

Table 62-18 (Cont.) Exponential Smoothing Settings

Setting Name	Setting Value	Description
`EXSM_BACKCAST_OUTPUT`	`EXSM_BACKCAST_OUTPUT_ENABLE` `EXSM_BACKCAST_OUTPUT_DISABLE`	This setting enables the user to optionally suppress the output of backcast values. Backcasts are the model estimates for historical data. See Backcasts in Time Series for information on backcasts. Suppressing the output of backcast values can provide a potentially large reduction in the memory and storage requirements for a partitioned ESM model with a huge number of partitions. The default value is `EXSM_BACKCAST_OUTPUT_ENABLE`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about ESM.
>
> https://github.com/oracle-samples/oracle-db-examples/tree/main/machine-learning/sql browse to the release folder and click the `oml4sql-time-series-exponential-smoothing.sql` example.

DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model

The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.

The settings listed in the following table specify the setting names and possible values for Generalized Linear Model. The Constant Value column specifies constants using the prefix `DBMS_DATA_MINING`. Alternatively, you can specify the corresponding string value from the String Value Equivalent column.

For Global settings, see DBMS_DATA_MINING — Global Settings.

For generic machine learning function settings, see DBMS_DATA_MINING — Machine Learning Functions.

Table 62-19 DBMS_DATA_MINING GLM Settings

Setting Name	Constant Value	String Value Equivalent	Description
GLMS_CONF_LEVEL	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	The confidence level for coefficient confidence intervals. The default confidence level is 0.95. Expression: TO_CHAR(0.98)
GLMS_FTR_GEN_METHOD	GLMS_FTR_GEN_QUADRATIC GLMS_FTR_GEN_CUBIC	GLMS_FTR_GEN_QUADRATIC GLMS_FTR_GEN_CUBIC	Whether feature generation is quadratic or cubic. When feature generation is enabled, the algorithm automatically chooses the most appropriate feature generation method based on the data.
GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_ENABLE GLMS_FTR_GENERATION_DISABLE	GLMS_FTR_GENERATION_ENABLE GLMS_FTR_GENERATION_DISABLE	Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled. **Note:** Feature generation can only be enabled when feature selection is also enabled.
GLMS_FTR_SEL_CRIT	GLMS_FTR_SEL_AIC GLMS_FTR_SEL_SBIC GLMS_FTR_SEL_RIC GLMS_FTR_SEL_ALPHA_INV	GLMS_FTR_SEL_AIC GLMS_FTR_SEL_SBIC GLMS_FTR_SEL_RIC GLMS_FTR_SEL_ALPHA_INV	Feature selection penalty criterion for adding a feature to the model. When feature selection is enabled, the algorithm automatically chooses the penalty criterion based on the data.
GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_ENABLE GLMS_FTR_SELECTION_DISABLE	GLMS_FTR_SELECTION_ENABLE GLMS_FTR_SELECTION_DISABLE	Whether or not feature selection is enabled for GLM. By default, feature selection is not enabled.
GLMS_MAX_FEATURES	An integer greater than 0 and less than or equal to 2000, represented as a character string	An integer greater than 0 and less than or equal to 2000, represented as a character string	When feature selection is enabled, this setting specifies the maximum number of features that can be selected for the final model. By default, the algorithm limits the number of features to ensure sufficient memory. Expression: TO_CHAR(200)
GLMS_PRUNE_MODEL	GLMS_PRUNE_MODEL_ENABLE GLMS_PRUNE_MODEL_DISABLE	GLMS_PRUNE_MODEL_ENABLE GLMS_PRUNE_MODEL_DISABLE	Prune enable or disable for features in the final model. Pruning is based on T-Test statistics for linear regression, or Wald Test statistics for logistic regression. Features are pruned in a loop until all features are statistically significant with respect to the full data. When feature selection is enabled, the algorithm automatically prunes based on the data.

Table 62-19 (Cont.) DBMS_DATA_MINING GLM Settings

Setting Name	Constant Value	String Value Equivalent	Description
GLMS_REFERENCE_CLASS_NAME	*target_value*	*target_value*	The target value used as the reference class in a binary logistic regression model. Probabilities are produced for the other class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.
GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	Enable or disable ridge regression. Ridge applies to both regression and classification machine learning functions. When ridge is enabled, prediction bounds are not produced by the PREDICTION_BOUNDS SQL function. **Note**: Ridge may only be enabled when feature selection is not specified, or has been explicitly disabled. If ridge regression and feature selection are both explicitly enabled, then an exception is raised.
GLMS_RIDGE_VALUE	An integer greater than 0 represented as a character string	An integer greater than 0 represented as a character string	The value of the ridge parameter. This setting is only used when the algorithm is configured to use ridge regression. If ridge regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm. Expression: TO_CHAR(5)
GLMS_ROW_DIAGNOSTICS	GLMS_ROW_DIAG_ENABLE GLMS_ROW_DIAG_DISABLE (default).	GLMS_ROW_DIAG_ENABLE GLMS_ROW_DIAG_DISABLE (default).	Enable or disable row diagnostics.
GLMS_CONV_TOLERANCE	The range is (0, 1) non-inclusive.	The range is (0, 1) non-inclusive.	Convergence Tolerance setting of the GLM algorithm The default value is system-determined.
GLMS_NUM_ITERATIONS	A positive integer	A positive integer	Maximum number of iterations for the GLM algorithm. The default value is system-determined.
GLMS_BATCH_ROWS	0 or a positive integer	0 or a positive integer	Number of rows in a batch used by the SGD solver. The value of this parameter sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 2000

Table 62-19 (Cont.) DBMS_DATA_MINING GLM Settings

Setting Name	Constant Value	String Value Equivalent	Description
GLMS_SOLVER	GLMS_SOLVER_SGD (StochasticGradient Descent) GLMS_SOLVER_CHOL (Cholesky) GLMS_SOLVER_QR GLMS_SOLVER_LBFGS_ADMM	GLMS_SOLVER_SGD (StochasticGradient Descent) GLMS_SOLVER_CHOL (Cholesky) GLMS_SOLVER_QR GLMS_SOLVER_LBFGS_ADMM	This setting allows the user to choose the GLM solver. The solver cannot be selected if GLMS_FTR_SELECTION setting is enabled. • GLMS_SOLVER_SGD: Optimizes generalized linear models by iteratively updating parameters using a subset of the data to minimize errors. • GLMS_SOLVER_CHOL: Solves generalized linear models using the Cholesky decomposition method, which provides a stable and efficient solution by transforming the right-hand of the equation into a lower triangular matrix and its conjugate transpose. • GLMS_SOLVER_QR: Utilizes the QR decomposition technique to solve generalized linear models, ensuring numerical stability and accuracy by decomposing the problem into an orthonormal matrix Q and upper triangular matrix R. • GLMS_SOLVER_LBFGS_ADMM: Combines L-BFGS, an approximation of the Broyden-Fletcher-Goldfarb-Shanno optimization algorithm, with ADMM for solving large-scale generalized linear model problems efficiently. The default value is system determined.
GLMS_SPARSE_SOLVER	GLMS_SPARSE_SOLVER_ENABLE GLMS_SPARSE_SOLVER_DISABLE (default).	GLMS_SPARSE_SOLVER_ENABLE GLMS_SPARSE_SOLVER_DISABLE (default).	This setting allows the user to use sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.

Table 62-19 (Cont.) DBMS_DATA_MINING GLM Settings

Setting Name	Constant Value	String Value Equivalent	Description
GLMS_LINK_FUNCTION	GLMS_IDENTITY_LINK GLMS_LOGIT_LINK GLMS_PROBIT_LINK GLMS_CLOGLOG_LINK GLMS_CAUCHIT_LINK	GLMS_IDENTITY_LINK GLMS_LOGIT_LINK GLMS_PROBIT_LINK GLMS_CLOGLOG_LINK GLMS_CAUCHIT_LINK	This setting allows the user to specify the link function for building a GLM model. The link functions are specific to the mining function. For classification, the following are applicable: • GLMS_LOGIT_LINK (default) • GLMS_PROBIT_LINK • GLMS_CLOGLOG_LINK • GLMS_CAUCHIT_LINK For regression, the following is applicable: • GLMS_IDENTITY_LINK (default)

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.
- DBMS_DATA_MINING — Solver Settings: ADMM
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.
- *Oracle Machine Learning for SQL Concepts*

See Also:

Oracle Machine Learning for SQL Concepts for information about GLM.

DBMS_DATA_MINING — Algorithm Settings: *k*-Means

The settings listed in the following table configure the behavior of the *k*-Means algorithm.

Table 62-20 k-Means Settings

Setting Name	Setting Value	Description
`KMNS_CONV_TOLERANCE`	`TO_CHAR(0<`*`numeric_expr`*`<1)`	Minimum Convergence Tolerance for *k*-Means. The algorithm iterates until the minimum Convergence Tolerance is satisfied or until the maximum number of iterations, specified in `KMNS_ITERATIONS`, is reached. Decreasing the Convergence Tolerance produces a more accurate solution but may result in longer run times. The default Convergence Tolerance is `0.001`.
`KMNS_DISTANCE`	`KMNS_COSINE` `KMNS_EUCLIDEAN`	Distance function for *k*-Means. The default distance function is `KMNS_EUCLIDEAN`.
`KMNS_ITERATIONS`	`TO_CHAR(`*`positive_numeric_expr`*`)`	Maximum number of iterations for *k*-Means. The algorithm iterates until either the maximum number of iterations is reached or the minimum Convergence Tolerance, specified in `KMNS_CONV_TOLERANCE`, is satisfied. The default number of iterations is `20`.
`KMNS_MIN_PCT_ATTR_SUPPORT`	`TO_CHAR(0<=`*`numeric_expr`*`<=1)`	Minimum percentage of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster. If the data is sparse or includes many missing values, a minimum support that is too high can cause very short rules or even empty rules. The default minimum support is `0.1`.
`KMNS_NUM_BINS`	`TO_CHAR(`*`numeric_expr`*`>0)`	Number of bins in the attribute histogram produced by *k*-means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value that have only one bin. The default number of histogram bins is `11`.
`KMNS_SPLIT_CRITERION`	`KMNS_SIZE` `KMNS_VARIANCE`	Split criterion for *k*-means. The split criterion controls the initialization of new *k*-Means clusters. The algorithm builds a binary tree and adds one new cluster at a time. When the split criterion is based on size, the new cluster is placed in the area where the largest current cluster is located. When the split criterion is based on the variance, the new cluster is placed in the area of the most spread-out cluster. The default split criterion is the `KMNS_VARIANCE`.
`KMNS_RANDOM_SEED`	Non-negative integer	This setting controls the seed of the random generator used during the *k*-Means initialization. It must be a non-negative integer value. The default is `0`.

Table 62-20 (Cont.) k-Means Settings

Setting Name	Setting Value	Description
KMNS_DETAILS	KMNS_DETAILS_NONE KMNS_DETAILS_HIERARCHY KMNS_DETAILS_ALL	This setting determines the level of cluster detail that are computed during the build. KMNS_DETAILS_NONE: No cluster details are computed. Only the scoring information is persisted. KMNS_DETAILS_HIERARCHY: Cluster hierarchy and cluster record counts are computed. This is the default value. KMNS_DETAILS_ALL: Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed.
KMNS_WINSORIZE	KMNS_WINSORIZE_ENABLE KMNS_WINSORIZE_DISABLE	To winorize data, enable or disable this parameter. Data is restricted in a window size of six standard deviations around the mean value when winsorize is enabled. This functionality can be used with AUTO_DATA_PREP turned ON and OFF. The values outside the range are replaced with the ends of the interval. Winsorize is not enabled by default. **Note:** Winsorize is only available when the KMNS_EUCLIDEAN distance function is used. An exception is raised if Winsorize is enabled and other distance functions are set.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

- For generic machine learning function settings related to Clustering, see DBMS_DATA_MINING — Machine Learning Functions.
- *Oracle Machine Learning for SQL Concepts* for information about *k*-Means

DBMS_DATA_MINING - Algorithm Settings: Multivariate State Estimation Technique - Sequential Probability Ratio Test

Settings that configure the training calibration behavior of the Multivariate State Estimation Technique - Sequential Probability Ratio Test algorithm.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.MSET_ADB_HEIGHT`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'MSET_ADB_HEIGHT'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-21 MSET-SPRT Settings

Setting Name	Setting Value	String Value Equivalent	Description
`MSET_ADB_HEIGHT`	A positive double	A positive double	Estimates the band within which signal values normally oscillate. The default value is `0.05`.
`MSET_ALERT_COUNT`	A positive integer	A positive integer	The number of the last *n* signals (the alert window) that should have passed the threshold to raise an alert. The alert count should be lower or equal to the alert window. The default value is `5`.
`MSET_ALERT_WINDOW`	A positive integer greater than or equal to `MSET_ALERT_COUNT`	A positive integer greater than or equal to `MSET_ALERT_COUNT`	The number of signals to consider in the SPRT hypothesis consolidation logic. The default value is `5`.
`MSET_ALPHA_PROB`	A positive double between 0 and 1	A positive double between 0 and 1	False Alarm Probability FAP (false positive). The default is `0.01`.
`MSET_BETA_PROB`	A positive double between 0 and 1	A positive double between 0 and 1	Missed Alarm Probability MAP (false negative). The default is `0.10`.
`MSET_HELDASIDE`	A positive integer	A positive integer	The approximate number of data rows used for MSET model calibration. You can use `ODMS_RANDOM_SEED` to change the held-aside sample. The default value is `10000`.
`MSET_MEMORY_VECTORS`	A positive integer	A positive integer	The default value is data driven.

Table 62-21 (Cont.) MSET-SPRT Settings

Setting Name	Setting Value	String Value Equivalent	Description
`MSET_PROJECTION_THRESHOLD`	A positive integer >0, <=10000	A positive integer >0, <=10000	Specifies whether to use random projections. When the number of sensors exceeds the setting value, random projections are used. To turn off random projections, set the threshold to a value that is equal to or greater than the number of sensors. The default value is `500`.
`MSET_STD_TOLERANCE`	A positive integer	A positive integer	The tolerance in standard deviations used in the SPRT calculation. The default value is `3`.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

DBMS_DATA_MINING — Algorithm Settings: Naive Bayes

The settings listed in the following table configure the behavior of the Naive Bayes algorithm.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.NABS_PAIRWISE_THRESHOLD`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'NABS_PAIRWISE_THRESHOLD'`.

Note:

The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-22 Naive Bayes Settings

Setting Name	Setting Value	String Value Equivalent	Description
`NABS_PAIRWISE_THRESHOLD`	A floating point number between 0 and 1, inclusive, expressed as a character string	A floating point number between 0 and 1, inclusive, expressed as a character string	Value of pairwise threshold for NB algorithm Default is `0`. Expression: `TO_CHAR(0.5)`

Table 62-22 (Cont.) Naive Bayes Settings

Setting Name	Setting Value	String Value Equivalent	Description
`NABS_SINGLETON_THRESHOLD`	A floating point number between 0 and 1, inclusive, expressed as a character string	A floating point number between 0 and 1, inclusive, expressed as a character string	Value of singleton threshold for NB algorithm Default value is `0`. Expression: `TO_CHAR(0.5)`

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about Naive Bayes

DBMS_DATA_MINING — Algorithm Settings: Neural Network

The settings listed in the following table configure the behavior of the Neural Network algorithm.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.NNET_SOLVER_ADAM`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'NNET_SOLVER_ADAM'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-23 DBMS_DATA_MINING Neural Network Settings

Setting Name	Constant Value	String Value Equivalents	Description
`NNET_SOLVER`	One of the following strings: `NNET_SOLVER_ADAM`	`NNET_SOLVER_ADAM`	Specifies the method of optimization. The default value is system determined. `NNET_SOLVER_ADAM`: Uses the Adam optimization method.

Table 62-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Constant Value	String Value Equivalents	Description
	NNET_SOLVER_LBFGS	NNET_SOLVER_LBFGS	Uses the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) optimization method.

NNET_ACTIVATIONS	One or more of the following strings:
	NNET_ACTIVATIONS_ARCTAN
		NNET_ACTIVATIONS_ARCTAN	Specifies the activation functions for the hidden layers. You can specify a single activation function, which is then applied to each hidden layer, or you can specify an activation function for each layer individually. Different layers can have different activation functions.
			To apply a different activation function to one or more of the layers, you must specify an activation function for each layer. The number of activation functions you specify must be consistent with the NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER values.
			For example, if you have three hidden layers, you could specify the use of the same activation function for all three layers with the following settings value:
			`('NNET_ACTIVATIONS', 'NNET_ACTIVATIONS_TANH')`
			The following settings value specifies a different activation function for each layer:
			`('NNET_ACTIVATIONS', '''NNET_ACTIVATIONS_TANH'', ''NNET_ACTIVATIONS_LOG_SIG'', ''NNET_ACTIVATIONS_ARCTAN''')`
			Note: You specify the different activation functions as strings within a single string. All quotes are single and two single quotes are used to escape a single quote in SQL statements and PL/SQL blocks.
			NNET_ACTIVATIONS_ARCTAN: Uses the arctangent activation function.
			The default value is NNET_ACTIVATIONS_LOG_SIG.

Table 62-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Constant Value	String Value Equivalents	Description
	NNET_ACTIVATIONS_BIPOLAR_SIG	NNET_ACTIVATIONS_BIPOLAR_SIG	Uses the bipolar sigmoid activation function.
	NNET_ACTIVATIONS_LINEAR	NNET_ACTIVATIONS_LINEAR	Uses the linear activation function.
	NNET_ACTIVATIONS_LOG_SIG	NNET_ACTIVATIONS_LOG_SIG	Uses the logistic sigmoid activation function.
	NNET_ACTIVATIONS_RELU	NNET_ACTIVATIONS_RELU	Uses the rectified linear unit activation function.
	NNET_ACTIVATIONS_TANH	NNET_ACTIVATIONS_TANH	Uses the hyperbolic tangent activation function.
NNET_HELDASIDE_MAX_FAIL	A positive integer	A positive integer	With NNET_REGULARIZER_HELDASIDE, the training process is stopped early if the network performance on the validation data fails to improve or remains the same for NNET_HELDASIDE_MAX_FAIL epochs in a row. The default value is 6.
NNET_HELDASIDE_RATIO	An integer greater than 0 and less than or equal to 1, represented as a character string	An integer greater than 0 and less than or equal to 1, represented as a character string	Define the held ratio for the held-aside method. The default value is 0.25. Expression: TO_CHAR(0.45)
NNET_HIDDEN_LAYERS	A positive integer	A positive integer	Defines the topology by the number of hidden layers. The default value is 1.
NNET_ITERATIONS	A positive integer	A positive integer	Specifies the maximum number of iterations in the Neural Network algorithm. For the DMSSET_NN_SOLVER_LBFGS solver, the default value is 200. For the DMSSET_NN_SOLVER_ADAM solver, the default value is 10000.
NNET_NODES_PER_LAYER	A positive integer or a list of positive integers	A positive integer or a list of positive integers	Defines the topology by the number of nodes per layer. Different layers can have different numbers of nodes. To specify the same number of nodes for each layer, you can provide a single value, which is then applied to each layer. To specify a different number of nodes for one or more layers, provide a list of comma-separated positive integers, one for each layer. For example, '10, 20, 5' for three layers. The setting values must be consistent with the NNET_HIDDEN_LAYERS value. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes > 50).

Table 62-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Constant Value	String Value Equivalents	Description
NNET_REG_LAMBDA	An integer greater than or equal to 0 represented as a character string	An integer greater than or equal to 0 represented as a character string	Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1. Expression: TO_CHAR(2)
NNET_REGULARIZER	One of the following strings: NNET_REGULARIZER_HELDASIDE	NNET_REGULARIZER_HELDASIDE	Regularization setting for Neural Network algorithm. NNET_REGULARIZER_HELDASIDE: Uses a held-aside method for regularization. If the total number of training rows is greater than 50000, the default is NNET_REGULARIZER_HELDASIDE.
	NNET_REGULARIZER_L2	NNET_REGULARIZER_L2	Applies L2 regularization, which penalizes the sum of squared weights.
	NNET_REGULARIZER_NONE	NNET_REGULARIZER_NONE	Disables regularization. If the total number of training rows is less than or equal to 50000, the default is NNET_REGULARIZER_NONE.
NNET_TOLERANCE	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001. Expression: TO_CHAR(0.00004)
NNET_WEIGHT_LOWER_BOUND	A real number	A real number	The setting specifies the lower bound of the region where weights are randomly initialized. NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND must be set together. Setting one and not setting the other raises an error. NNET_WEIGHT_LOWER_BOUND must not be greater than NNET_WEIGHT_UPPER_BOUND. The default value is -sqrt(6/(l_nodes+r_nodes)). The value of l_nodes for: • input layer dense attributes is (1+number of dense attributes) • input layer sparse attributes is number of sparse attributes • each hidden layer is (1+number of nodes in that hidden layer) The value of r_nodes is the number of nodes in the layer that the weight is connecting to.

Table 62-23 (Cont.) DBMS_DATA_MINING Neural Network Settings

Setting Name	Constant Value	String Value Equivalents	Description
NNET_WEIGHT_UPPER_BOUND	A real number	A real number	This setting specifies the upper bound of the region where weights are initialized. It should be set in pairs with NNET_WEIGHT_LOWER_BOUND and its value must not be smaller than the value of NNET_WEIGHT_LOWER_BOUND. If not specified, the values of NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND are system determined. The default value is sqrt(6/(l_nodes+r_nodes)). See NNET_WEIGHT_LOWER_BOUND.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

See Also:

Oracle Machine Learning for SQL Concepts for information about Neural Network.

DBMS_DATA_MINING — Algorithm Settings: Non-Negative Matrix Factorization

The settings listed in the following table configure the behavior of the Non-negative Matrix Factorization algorithm.

The **Constant Value** column specifies constants using the prefix DBMS_DATA_MINING. For example, DBMS_DATA_MINING.NMFS_NONNEG_SCORING_ENABLE. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the DBMS_DATA_MINING prefix, in single quotes. For example, 'NMFS_NONNEG_SCORING_ENABLE'.

Note:

The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

You can query the data dictionary view *_MINING_MODEL_SETTINGS (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Database Reference* for information about *_MINING_MODEL_SETTINGS.

Table 62-24 NMF Settings

Setting Name	Constant Value	String Value Equivalent	Description
NMFS_CONV_TOLERANCE	A floating point number between 0 and 0.5 expressed as a character string	A floating point number between 0 and 0.5 expressed as a character string	Convergence tolerance for NMF algorithm Default is 0.05 Expression: TO_CHAR(0.02)
NMFS_NONNEGATIVE_SCORING	NMFS_NONNEG_SCORING_ENABLE	NMFS_NONNEG_SCORING_ENABLE	Whether negative numbers should be allowed in scoring results. When set to NMFS_NONNEG_SCORING_ENABLE, negative feature values will be replaced with zeros. Default is NMFS_NONNEG_SCORING_ENABLE
	NMFS_NONNEG_SCORING_DISABLE	NMFS_NONNEG_SCORING_DISABLE	When set to NMFS_NONNEG_SCORING_DISABLE, negative feature values will be allowed.
NMFS_NUM_ITERATIONS	An integer between 1 to 500, inclusive, represented as a character string	An integer between 1 to 500, inclusive, represented as a character string	Number of iterations for NMF algorithm Default is 50 Expression: TO_CHAR(80)
NMFS_RANDOM_SEED	An integer represented as a character string	An integer represented as a character string	Random seed for NMF algorithm. Default is -1. Expression: TO_CHAR(2)

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about NMF

DBMS_DATA_MINING — Algorithm Settings: O-Cluster

The settings in the table configure the behavior of the O-Cluster algorithm.

The **Constant Value** column specifies constants using the prefix DBMS_DATA_MINING. For example, DBMS_DATA_MINING.OCLT_SENSITIVITY. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'OCLT_SENSITIVITY'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-25 O-CLuster Settings

Setting Name	Constant Value	String Value Equivalent	Description
`OCLT_SENSITIVITY`	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Default is `0.5`. Example: `TO_CHAR(0.9)`

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about O-Cluster

DBMS_DATA_MINING — Algorithm Settings: Random Forest

These settings configure the behavior of the Random Forest algorithm. Random Forest makes use of the Decision Tree settings to configure the construction of individual trees.

The **Constant Value** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.RFOR_MTRY`. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'RFOR_MTRY'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-26 Random Forest Settings

Setting Name	Constant Value	String Value Equivalent	Description
RFOR_MTRY	a number >= 0	a number >= 0	Size of the random subset of columns to be considered when choosing a split at a node. For each node, the size of the pool remains the same, but the specific candidate columns change. The default is half of the columns in the model signature. The special value 0 indicates that the candidate pool includes all columns.
RFOR_NUM_TREES	1<= *a number* <=65535	1<= *a number* <=65535	Number of trees in the forest Default is 20.
RFOR_SAMPLING_RATIO	0< *a fraction*<=1	0< *a fraction*<=1	Fraction of the training data to be randomly sampled for use in the construction of an individual tree. The default is half of the number of rows in the training data.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.
- DBMS_DATA_MINING — Algorithm Settings: Decision Tree
  These settings configure the behavior of the Decision Tree algorithm. Note that the Decision Tree settings are also used to configure the behavior of Random Forest as it constructs each individual decision tree.

See Also:

Oracle Machine Learning for SQL Concepts for information about Random Forest

DBMS_DATA_MINING — Algorithm Constants and Settings: Singular Value Decomposition

The following settings configure the behavior of the Singular Value Decomposition algorithm.

Table 62-27 Singular Value Decomposition Settings

Setting Name	Constant Value	String Value Equivalent	Description
`SVDS_U_MATRIX_OUTPUT`	`SVDS_U_MATRIX_ENABLE`	`SVDS_U_MATRIX_ENABLE`	Indicates whether or not to persist the **U** Matrix produced by SVD. The U matrix in SVD has as many rows as the number of rows in the build data. To avoid creating a large model, the U matrix is persisted only when `SVDS_U_MATRIX_OUTPUT` is enabled. When `SVDS_U_MATRIX_OUTPUT` is enabled, the build data must include a case ID. If no case ID is present and the U matrix is requested, then an exception is raised. Default is `SVDS_U_MATRIX_DISABLE`.
	`SVDS_U_MATRIX_DISABLE`	`SVDS_U_MATRIX_DISABLE`	Does not persist the U Matrix.
`SVDS_SCORING_MODE`	`SVDS_SCORING_SVD`	`SVDS_SCORING_SVD`	Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections will be the same as the U matrix. Default is `SVDS_SCORING_SVD`.
	`SVDS_SCORING_PCA`	`SVDS_SCORING_PCA`	When the build data is scored with PCA, the projections will be the product of the U and S matrices.

Table 62-27 (Cont.) Singular Value Decomposition Settings

Setting Name	Constant Value	String Value Equivalent	Description
SVDS_SOLVER	SVDS_SOLVER_TSSVD	SVDS_SOLVER_TSSVD	This setting indicates the solver to be used for computing SVD of the data. In the case of PCA, the solver setting indicates the type of SVD solver used to compute the PCA for the data. When this setting is not specified the solver type selection is data driven. If the number of attributes is greater than 3240, then the default wide solver is used. Otherwise, the default narrow solver is selected. The following are the group of solvers: • Narrow data solvers: for matrices with up to 11500 attributes (TSEIGEN) or up to 8100 attributes (TSSVD). • Wide data solvers: for matrices up to 1 million attributes. For narrow data solvers: • Tall-Skinny SVD uses QR computation TSVD (SVDS_SOLVER_TSSVD) • Tall-Skinny SVD uses eigenvalue computation, TSEIGEN (SVDS_SOLVER_TSEIGEN), is the default solver for narrow data. For wide data solvers: • Stochastic SVD uses QR computation SSVD (SVDS_SOLVER_SSVD), is the default solver for wide data solvers. • Stochastic SVD uses eigenvalue computations, STEIGEN (SVDS_SOLVER_STEIGEN).
	SVDS_SOLVER_TSEIGEN	SVDS_SOLVER_TSEIGEN	Tall-Skinny SVD using eigenvalue computation for matrices with up to 11500 attributes. This is the default solver for narrow data.
	SVDS_SOLVER_SSVD	SVDS_SOLVER_SSVD	Stochastic SVD using QR computation for matrices with up to 1 million attributes. This is the default solver for wide data.
	SVDS_SOLVER_STEIGEN	SVDS_SOLVER_STEIGEN	Stochastic SVD using eigenvalue computations for matrices with up to 1 million attributes.
SVDS_TOLERANCE	Range [0, 1]	Range [0, 1]	This setting is used to prune features. Define the minimum value the eigenvalue of a feature as a share of the first eigenvalue to not to prune. Default value is data driven.
SVDS_RANDOM_SEED	Range [0 - 4,294,967,296]	Range [0 - 4,294,967,296]	The random seed value is used for initializing the sampling matrix used by the Stochastic SVD solver. The default is 0. The SVD Solver must be set to SSVD or STEIGEN.

Table 62-27 (Cont.) Singular Value Decomposition Settings

Setting Name	Constant Value	String Value Equivalent	Description
SVDS_OVER_SAMPLING	Range [1, 5000].	Range [1, 5000].	This setting is configures the number of columns in the sampling matrix used by the Stochastic SVD solver. The number of columns in this matrix is equal to the requested number of features plus the oversampling setting. The SVD Solver must be set to SSVD or STEIGEN.
SVDS_POWER_ITERATIONS	Range [0, 20].	Range [0, 20].	The power iteration setting improves the accuracy of the SSVD solver. The default is 2. The SVD Solver must be set to SSVD or STEIGEN.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts

DBMS_DATA_MINING — Algorithm Settings: Support Vector Machine

The settings listed in the following table configure the behavior of the Support Vector Machine algorithm.

The **Constant Value** column specifies constants using the prefix DBMS_DATA_MINING. For example, DBMS_DATA_MINING.SVMS_GAUSSIAN. Alternatively, you can specify the corresponding string value from the **String Value Equivalent** column without the DBMS_DATA_MINING prefix, in single quotes. For example, 'SVMS_GAUSSIAN'.

Note:

The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

Table 62-28 SVM Settings

Setting Name	Constant Value	String Value Equivalent	Description
SVMS_COMPLEXITY_FACTOR	An integer greater than 0 represented as a character string	An integer greater than 0 represented as a character string	Regularization setting that balances the complexity of the model against model robustness to achieve good generalization on new data. SVM uses a data-driven approach to finding the complexity factor. Value of complexity factor for SVM algorithm (both classification and regression). Default value estimated from the data by the algorithm. Expression: TO_CHAR(20)
SVMS_CONV_TOLERANCE	An integer greater than 0 represented as a character string	An integer greater than 0 represented as a character string	Convergence tolerance for SVM algorithm. Default is 0.0001. Expression: TO_CHAR(0.005)
SVMS_EPSILON	An integer greater than 0 represented as a character string	An integer greater than 0 represented as a character string	Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data. Value of epsilon factor for SVM regression. Default is 0.1. Expression: TO_CHAR(0.5)
SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN	SVMS_GAUSSIAN	Kernel for Support Vector Machine. Linear or Gaussian. SVMS_GAUSSIAN: Uses the Gaussian kernel for SVM. The default value is SVMS_LINEAR.
	SVMS_LINEAR	SVMS_LINEAR	Uses the Linear kernel for SVM. This is the default option.
SVMS_OUTLIER_RATE	A floating point number between 0 and 1 expressed as a character string	A floating point number between 0 and 1 expressed as a character string	The desired rate of outliers in the training data. Valid for One-Class SVM models only (anomaly detection). Default is 0.01. Expression: TO_CHAR(0.04)
SVMS_STD_DEV	An integer greater than 0 represented as a character string	An integer greater than 0 represented as a character string	Controls the spread of the Gaussian kernel function. SVM uses a data-driven approach to find a standard deviation value that is on the same scale as distances between typical cases. Value of standard deviation for SVM algorithm. This is applicable only for Gaussian kernel. Default value estimated from the data by the algorithm. Expression: TO_CHAR(6)

Table 62-28 (Cont.) SVM Settings

Setting Name	Constant Value	String Value Equivalent	Description
SVMS_NUM_ITERATIONS	A positive integer	A positive integer	This setting sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.
SVMS_NUM_PIVOTS	Range [1; 10000]	Range [1; 10000]	This setting sets an upper limit on the number of pivots used in the Incomplete Cholesky decomposition. It can be set only for non-linear kernels. The default value is 200.
SVMS_BATCH_ROWS	A positive integer	A positive integer	This setting applies to SVM models with linear kernel. This setting sets the size of the batch for the SGD solver. An input of 0 triggers a data driven batch size estimate. The default is 20000.
SVMS_REGULARIZER	SVMS_REGULARIZER_L1	SVMS_REGULARIZER_L1	This setting controls the type of regularization that the SGD SVM solver uses. The setting can be used only for linear SVM models. The default is system determined because it depends on the potential model size. SVMS_REGULARIZER_L1: Uses L1 regularization.
	SVMS_REGULARIZER_L2	SVMS_REGULARIZER_L2	Uses L2 regularization.
SVMS_SOLVER	SVMS_SOLVER_SGD (Sub-Gradient Descend)	SVMS_SOLVER_SGD (Sub-Gradient Descend)	Enables to choose the SVM solver. The SGD solver cannot be selected if the kernel is non-linear. The default value is system determined. SVMS_SOLVER_SGD: Uses Sub-Gradient Descent solver.
	SVMS_SOLVER_IPM (Interior Point Method)	SVMS_SOLVER_IPM (Interior Point Method)	Uses Interior Point Method solver.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

See Also:

Oracle Machine Learning for SQL Concepts for information about SVM

DBMS_DATA_MINING — Algorithm Settings: XGBoost

Settings that configure the behavior of the XGBoost gradient boosting algorithm.

The **Constant Name** column specifies constants using the prefix `DBMS_DATA_MINING`. For example, `DBMS_DATA_MINING.xgboost_booster`. Alternatively, you can specify the corresponding string value from the **String Name Equivalent** column without the `DBMS_DATA_MINING` prefix, in single quotes. For example, `'booster'`.

> **Note:**
>
> The distinction between **Constant Value** and **String Value Equivalent** for this algorithm is applicable to Oracle Database 19*c* and Oracle Database 21*c*.

The XGBoost settings are case sensitive. Enter the settings as they appear in the settings table. These settings match the XGBoost settings available in open source. OML4SQL XGBoost is based on the 1.7.4 version of XGBoost.
For Global settings, see DBMS_DATA_MINING — Global Settings.

For generic machine learning technique settings, see DBMS_DATA_MINING — Machine Learning Functions.

Table 62-29 General Settings

Constant Name	String Name Equivalent	Setting Value	Description

`xgboost_booster`	`booster`	A string that is one of the following:
		`dart`
		`gblinear`
		`gbtree`
			The booster to use:
			• `dart`
			• `gblinear`
			• `gbtree`
			The `dart` and `gbtree` boosters use tree-based models whereas `gblinear` uses linear functions.
			The default value is `gbtree`.

`xgboost_num_round`	`num_round`	A non-negative integer.	The number of rounds for boosting. The default value is `10`.

Table 62-30 Settings for Tree Boosting

Constant Name	String Name Equivalent	Setting Value	Description
`xgboost_alpha`	`alpha`	A non-negative number	L1 regularization term on weights. Increasing this value makes the model more conservative. The default value is `0`.
`xgboost_colsample_bylevel`	`colsample_bylevel`	A number in the range (0, 1]	Subsample ratio of columns for each split, in each level. Subsampling occurs each time a new split is made. This parameter has no effect when `tree_method` is set to `hist`. The default value is `1`.

Table 62-30 (Cont.) Settings for Tree Boosting

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_colsample_bynode	colsample_bynode	A number in the range (0, 1]	The subsample ratio of columns for each node (split). Subsampling occurs once every time a new split is evaluated. Columns are subsampled from the set of columns chosen for the current level. The default value is 1.
xgboost_colsample_bytree	colsample_bytree	A number in the range (0, 1]	Subsample ratio of columns when constructing each tree. Subsampling occurs once in every boosting iteration. The default value is 1.
xgboost_eta	eta	A number in the range [0, 1]	Step-size shrinkage used in the update step to prevent overfitting. After each boosting step, eta shrinks the feature weights to make the boosting process more conservative. The default value is 0.3.
xgboost_gamma	gamma	A number in the range [0, ∞]	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma value is, the more conservative the algorithm is. The default value is 0.
xgboost_grow_policy	grow_policy	A string; one of the following: • depthwise • lossguide	Controls the way new nodes are added to the tree: • depthwise splits at nodes closest to the root • lossguide splits at nodes with the highest loss change Valid only if tree_method is set to hist. The default value is depthwise.
xgboost_interaction_constraints	interaction_constraints	[[x0,x1,x2],[x0,x4],[x5,x6]] where xn are feature names or columns	This setting specifies permitted interactions in the model. Specify the constrains in the form of a nested list where each inner list is a group of features (column names) that are allowed to interact with each other. If a single column is passed in the interactions then, the input is ignored. Here, features x0, x1, and x2 are allowed to interact with each other but with no other feature. Similarly, x0 and x4 are allowed to interact with each other but with no other feature and so on. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.
xgboost_lambda	lambda	A non-negative number	L2 regularization term on weights. The default value is 1.

Table 62-30 (Cont.) Settings for Tree Boosting

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_max_bin	max_bin	A non-negative integer	Maximum number of discrete bins to bucket continuous features. Increasing this number improves the optimality of splits at the cost of higher computation time. This parameter is valid only when tree_method is set to hist. The default value is 256.
xgboost_max_delta_step	max_delta_step	A number in the range [0, ∞]	Maximum delta step allowed for each leaf output. Setting this to a positive value can help make the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when the class is extremely imbalanced. Setting it to value from 1 to 10 might help control the update. The default value is 0, which means there is no constraint.
xgboost_max_depth	max_depth	An integer in the range [0, ∞]	Maximum depth of a tree. Increasing this value makes the model more complex and more likely to overfit. Setting this to 0 indicates no limit. **Note:** You must set a max_depth limit when the grow_policy setting is depthwise. The default value is 6.
xgboost_max_leaves	max_leaves	A non-negative number	Maximum number of nodes to add. Use this setting only when grow_policy is set to lossguide. The default value is 0.
xgboost_min_child_weight	min_child_weight	A number in the range [0, ∞]	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with a sum of instance weight less than min_child_weight, then the building process stops partitioning. In a linear regression task, this corresponds to the minimum number of instances needed in each node. The larger min_child_weight is, the more conservative the algorithm is. The default value is 1.

Table 62-30 (Cont.) Settings for Tree Boosting

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_decrease_constraints	num_parallel_tree	[x4,x5]	This setting specifies the features (column names) that must obey decreasing constraint. The feature names are separated by a comma. For example, setting value 'x4,x5' sets decreasing constraint on features x4 and x5. This setting applies to numeric columns and 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.
xgboost_increase_constraints	increase_constraints	[x0,x3]	This setting specifies the features (column names) that must obey increasing constraint. The feature names are separated by a comma. For example, setting value 'x0,x3' sets increasing constraint on features x0 and x3. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.
xgboost_num_parallel_tree	num_parallel_tree	A non-negative integer	Number of parallel trees constructed during each iteration. Use this option to support a boosted random forest. The default value is `1`.
xgboost_scale_pos_weight	scale_pos_weight	A non-negative number	Controls the balance of positive and negative weights, which is useful for unbalanced classes. A typical value to consider: `sum(negative cases) / sum(positive cases)`. The default value is `1`.
xgboost_sketch_eps	sketch_eps	A number in the range (0, 1)	Increases enumeration accuracy. Valid only for the approximate greedy tree method. Compared to directly selecting the number of bins, this setting comes with a theoretical guarantee with sketch accuracy. You usually do not need to change this setting, but you might consider setting a lower number for more accurate enumeration. The default value is `0.03`.
xgboost_subsample	subsample	A number in the range (0, 1]	Subsample ratio of the training instances. A setting of 0.5 means that XGBoost randomly samples half of the training data prior to growing trees, which prevents overfitting. Subsampling occurs once in every boosting iteration. The default value is `1`.

Table 62-30 (Cont.) Settings for Tree Boosting

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_tree_method	tree_method	A string that is one of the following: • approx • auto • exact • hist	Tree construction algorithm used in XGBoost: • approx: Approximate greedy algorithm using sketching and histogram. • auto: Use a heuristic to choose the faster algorithm: – For a small to medium sized data set, uses the exact greedy algorithm. – For a very large data set, uses the approximate greedy algorithm. • exact: Exact greedy algorithm. • hist: Fast histogram optimized approximate greedy algorithm; uses some performance improvements such as bins caching. The default value is auto.
xgboost_updater	updater	A comma-separated string; one or more of the following: • grow_colmaker • grow_histmaker • grow_skmaker • grow_quantile_histmaker • prune • sync	Defines the sequence of tree updaters to run, which provides a modular way to construct and to modify the trees. This is an advanced parameter that is usually set automatically, depending on some other parameters. However, you can also explicitly specify a settting. The setting values are: • grow_colmaker: Non-distributed column-based construction of trees. • grow_histmaker: Distributed tree construction with row-based data splitting based on a global proposal of histogram counting. • grow_skmaker: Uses the approximate sketching algorithm. • grow_quantile_histmaker: Grow tree using quantized histogram. • prune: Prunes the splits where loss < min_split_loss (or gamma). • sync: Synchronizes trees in all distributed nodes.

Table 62-31 Settings for the Dart Booster

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_one_drop	one_drop	A number that is 0 or 1	When set to 1, at least one tree is always dropped during the dropout. When set to 0, at least one tree is not always dropped during the dropout. The default value is 0.

Table 62-31 (Cont.) Settings for the Dart Booster

Constant Name	String Name Equivalent	Setting Value	Description
`xgboost_normalize_type`	`normalize_type`	A string; either: • `forest` • `tree`	Type of normalization algorithm: • `forest`: New trees have the same weight as the sum of the dropped trees (forest): – The weight of new trees is `1 / (1 + learning_rate)` – Dropped trees are scaled by a factor of `1 / (1 + learning_rate)` • `tree`: New trees have the same weight as dropped trees: – The weight of new trees is `1 / (k + learning_rate)` – Dropped trees are scaled by a factor of `k / (k + learning_rate)` The default value is `tree`.
`xgboost_rate_drop`	`rate_drop`	A number in the range [0.0, 1.0]	Dropout rate (a fraction of the previous trees to drop during the dropout). The default value is `0.0`.
`xgboost_sample_type`	`sample_type`	A string; either: • `uniform` • `weighted`	Type of sampling algorithm: • `uniform`: Dropped trees are selected uniformly • `weighted`: Dropped trees are selected in proportion to weight The default value is `uniform`.
`xgboost_skip_drop`	`skip_drop`	A number in the range [0.0, 1.0]	Probability of skipping the dropout procedure during a boosting iteration. If a dropout is skipped, new trees are added in the same manner as `gbtree`. A non-zero `skip_drop` has higher priority than `rate_drop` or `one_drop`. The default value is `0.0`.

Table 62-32 Settings for the Linear Booster

Constant Name	String Name Equivalent	Setting Value	Description
`xgboost_alpha`	`alpha`	A non-negative number	L1 regularization term on weights, normalized to the number of training examples. Increasing this value makes the model more conservative. The default value is `0`.

Table 62-32 (Cont.) Settings for the Linear Booster

<table>
<tr><th>Constant Name</th><th>String Name Equivalent</th><th>Setting Value</th><th>Description</th></tr>
<tr><td><code>xgboost_feature_selector</code></td><td><code>feature_selector</code></td><td>A string that is one of the following:
<code>cyclic</code>
<code>greedy</code>
<code>random</code>
<code>shuffle</code>
<code>thrifty</code></td><td>Feature selection and ordering method:
<code>cyclic</code>: Deterministic selection by cycling through the features one at a time.
<code>greedy</code>: Selects the coordinate with the greatest gradient magnitude. This method:
Has <code>O(num_feature^2)</code> complexity
Is fully deterministic
Allows restricting the selection to the <code>top_k</code> features per group with the largest magnitude of univariate weight change, by setting the <code>top_k</code> parameter; doing so reduces the complexity to <code>O(num_feature*top_k)</code>.
<code>random</code>: A random (with replacement) coordinate selector.
<code>shuffle</code>: Similar to <code>cyclic</code> but with random feature shuffling prior to each update.
<code>thrifty</code>: Thrifty, approximately-greedy feature selector. Prior to cyclic updates, reorders features in descending magnitude of their univariate weight changes. This operation is multithreaded and is a linear complexity approximation of the quadratic greedy selection. Restricts the selection per group to the <code>top_k</code> features with the largest magnitude of univariate weight change.
The default value is <code>cyclic</code>.</td></tr>
<tr><td><code>xgboost_lambda</code></td><td><code>lambda</code></td><td>A non-negative number</td><td>L2 regularization term on weights, normalized to the number of training examples. Increasing this value makes the model more conservative.
The default value is <code>0</code>.</td></tr>
<tr><td><code>xgboost_top_k</code></td><td><code>top_k</code></td><td>A non-negative integer</td><td>Number of top features to select for the <code>greedy</code> or <code>thrifty</code> feature selector. The value of 0 uses all of the features.
The default value is <code>0</code>.</td></tr>
<tr><td><code>xgboost_updater</code></td><td><code>updater</code></td><td>A string that is one of the following:
<code>coord_descent</code>
<code>shotgun</code></td><td>Algorithm to fit the linear model:
<code>coord_descent</code>: Ordinary coordinate descent algorithm; multithreaded but still produces a deterministic solution.
<code>shotgun</code>: Parallel coordinate descent algorithm based on the <code>shotgun</code> algorithm; uses "hogwild" parallelism and therefore produces a nondeterministic solution on each run.
The default value is <code>shotgun</code>.</td></tr>
</table>

Table 62-33 Settings for Tweedie Regression

Constant Name	String Name Equivalent	Setting Value	Description
xgboost_tweedie_variance_power	tweedie_variance_power	A number in the range (1, 2)	Controls the variance of the Tweedie distribution var(y) ~ E(y)^tweedie_variance_power. A setting closer to 1 shifts towards a Poisson distribution. A setting closer to 2 shifts towards a gamma distribution. The default value is 1.5.

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

Table 62-34 Settings for Learning Tasks

Setting Name	String Name Equivalent	Setting Value	Description
`xgboost_objective`	`objective`	For a classification model, a string that is one of the following: • `binary:hinge` • `binary:logistic` • `multi:softmax` • `multi:softprob` For a regression model, a string that is one of the following: • `binary:logitraw` • `count:poisson` • `rank:map` • `rank:ndcg` • `rank:pairwise` • `reg:gamma` • `reg:logistic` • `reg:tweedie` • `survival:aft` • `survival:cox` • `reg:squarederror` • `reg:squaredlogerror`	**Settings for a Classification model:** • `binary:hinge`: Hinge loss for binary classification. This setting makes predictions of 0 or 1, rather than producing probabilities. • `binary:logistic`: Logistic regression for binary classification. The output is the probability. • `multi:softmax`: Performs multiclass classification using the `softmax` objective; you must also set `num_class(`*`number_of_classes`*`)`. • `multi:softprob`: : Same as `softmax`, except the output is a vector of `ndata * nclass`, which can be further reshaped to an `ndata * nclass` matrix. The result contains the predicted probability of each data point belonging to each class. The default `objective` value for classification is `multi:softprob`. **Settings for a Regression model:** • `binary:logitraw`: Logistic regression for binary classification; the output is the score before logistic transformation. • `count:poisson`: Poisson regression for count data; the output is the mean of the Poisson distribution. The `max_delta_step` value is set to 0.7 by default in Poisson regression to safeguard optimization. • `rank:map`: Using `LambdaMART`, performs list-wise ranking in which the Mean Average Precision (MAP) is maximized. • `rank:ndcg`: Using `LambdaMART`, performs list-wise ranking in which the Normalized Discounted Cumulative Gain (NDCG) is maximized. • `rank:pairwise`: Performs ranking by minimizing the pairwise loss. • `reg:gamma`: Gamma regression with log-link; the output is the mean of the gamma distribution. This setting might be useful for any outcome that might be gamma-distributed, such as modeling insurance claims severity. • `reg:logistic`: Logistic regression. • `reg:tweedie`: Tweedie regression with log-link. This setting might be useful for any outcome that might be Tweedie-distributed, such as modeling total loss in insurance. • `survival:aft`: Applies the Accelerated Failure Time (AFT) model for censored survival time data. When you select this

Table 62-34 (Cont.) Settings for Learning Tasks

Setting Name	String Name Equivalent	Setting Value	Description
			option, `eval_metric` uses `aft-nloglik` as the default value. • `survival:cox`: Cox regression for right-censored survival time data (negative values are considered right-censored). Predictions are returned on the hazard ratio scale (that is, as `HR = exp(marginal_prediction)` in the proportional hazard function `h(t) = h0(t) * HR)`. • `reg:squarederror`: Regression with squared loss. • `reg:squaredlogerror`: Regression with squared log loss. All input labels must be greater than -1. The default `objective` value for regression is `reg:squarederror`.
`xgboost_aft_loss_distribution`	`aft_loss_distribution`	[normal, logistic, extreme]	Specifies the distribution of the Z term in the AFT model. It specifies the Probabilty Density Function used by `survival:aft` objective and `aft-nloglik` evaluation metric. The default value is `normal`.
`xgboost_aft_loss_distribution_scale`	`aft_loss_distribution_scale`	A positive number	Specifies the scaling factor σ, which scales the size of Z term in the AFT model. The default value is `1`.
`xgboost_aft_right_bound_column_name`	`aft_right_bound_column_name`	*column_name*	Specifies the column containing the right bounds of the labels for an AFT model. You cannot select this parameter for a non-AFT model. **Note:** Oracle Machine Learning does not support `BOOLEAN` values for this setting.
`xgboost_base_score`	`base_score`	A number	Initial prediction score of all instances, global bias. For a sufficient number of iterations, changing this value does not have much effect. The default value is `0.5`.

Table 62-34 (Cont.) Settings for Learning Tasks

Setting Name	String Name Equivalent	Setting Value	Description
`xgboost_eval_metric`	`eval_metric`	A comma-separated string; one or more of the following: • `aft-nloglik` • `auc` • `aucpr` • `cox-nloglik` • `error` • `error@t` • `gamma-deviance` • `gamma-nloglik` • `logloss` • `mae` • `map` • `map@n` • `merror` • `mlogloss` • `ndcg` • `ndcg@n` • `poisson-nloglik` • `rmse` • `tweedie-nloglik@rho` • `ndcg-` • `map-` • `rmsle`	Evaluation metrics for validation data. You can specify one or more of these evaluation metrics: • `aft-nloglik`: Sets the `eval_metric` to negative log likelihood of AFT model. • `auc`: Area under the curve. • `aucpr`: Area under the PR curve. • `cox-nloglik`: Negative partial log-likelihood for Cox proportional hazards regression. • `error`: Binary classification error rate, calculated as the number of wrong cases divided by the number of all cases. For the predictions, the evaluation regards the instances with a prediction value larger than 0.5 as positive instances, and the others as negative instances. • `error@t`: You can specify a binary classification threshold value other than 0.5 by specifying a numerical value *t*; for example, `error@0.8`. • `gamma-deviance`: Residual deviance for `gamma` regression. • `gamma-nloglik`: Negative log-likelihood for `gamma` regression. • `logloss`: Negative log-likelihood. • `mae`: Mean absolute error. • `map`: Mean average precision. • `map@n`: Assigns the integer *n* as the cut-off value for the top positions in the lists for evaluation. • `merror`: Multiclass classification error rate calculated as the number of wrong cases divided by the number of all cases; the objective must be `multi:softprob` or `multi:softmax`. • `mlogloss`: Multiclass `logloss`; the objective must be `multi:softprob` or `multi:softmax`. • `ndcg`: Normalized Discounted Cumulative Gain. • `ndcg@n`: Assigns the integer *n* as the cut-off value for the top positions in the lists for evaluation. • `poisson-nloglik`: Negative log-likelihood for Poisson regression • `rmse`: Root Mean Square Error. • `tweedie-nloglik@rho`: Negative log-likelihood for Tweedie regression (at a specified value `rho` of the `tweedie_variance_power` parameter);

Table 62-34 (Cont.) Settings for Learning Tasks

Setting Name	String Name Equivalent	Setting Value	Description
			rho must be a number in the range (1, 2); for example, tweedie-nloglik@1.8. • ndcg- and map-: In XGBoost, NDCG and MAP will evaluate the score of a list without any positive samples as 1. By adding "-" in the evaluation metric XGBoost will evaluate these score as 0 to be consistent under some conditions. • rmsle: It is root mean square log error. This is the default metric of reg:squaredlogerror objective. This metric reduces errors generated by outliers in dataset. But because log function is employed, rmsle might output nan when prediction value is less than -1. A default metric is assigned according to the objective: • error for classification • mean average precision for ranking • rmse for regression
xgboost_seed	seed	A non-negative integer	Random number seed. The default value is 0.

Related Topics

- DBMS_DATA_MINING — Machine Learning Functions
  A machine learning **function** refers to the methods for solving a given class of machine learning problems.
- DBMS_DATA_MINING — Global Settings
  The configuration settings in this table are applicable to any type of model, but are currently only implemented for specific algorithms.

> **See Also:**
>
> https://github.com/oracle/oracle-db-examples/tree/master/machine-learning/sql/, select the release, and browse for an example of XGBoost.

DBMS_DATA_MINING — Solver Settings

Oracle Machine Learning for SQL algorithms can use different solvers. Solver settings can be provided at build time in the settings table.

Related Topics

- DBMS_DATA_MINING - Solver Settings: Adam
  These settings configure the behavior of the Adaptive Moment Estimation (Adam) solver.

- DBMS_DATA_MINING — Solver Settings: ADMM
  The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.
- DBMS_DATA_MINING — Solver Settings: LBFGS
  The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

DBMS_DATA_MINING - Solver Settings: Adam

These settings configure the behavior of the Adaptive Moment Estimation (Adam) solver.

Neural Network models use these settings.

Table 62-35 DBMS_DATA_MINING Adam Settings

Setting Name	Setting Value	Description
`ADAM_ALPHA`	A non-negative double precision floating point number in the interval (0; 1]	The learning rate for Adam. The default value is `0.001`.
`ADAM_BATCH_ROWS`	A positive integer	The number of rows per batch. The default value is `10000`.
`ADAM_BETA1`	A positive double precision floating point number in the interval [0; 1)	The exponential decay rate for the 1st moment estimates. The default value is `0.9`.
`ADAM_BETA2`	A positive double precision floating point number in the interval [0; 1)	The exponential decay rate for the 2nd moment estimates. The default value is `0.99`.
`ADAM_GRADIENT_TOLERANCE`	A positive double precision floating point number	The gradient infinity norm tolerance for Adam. The default value is `1E-9`.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.

DBMS_DATA_MINING — Solver Settings: ADMM

The settings listed in the following table configure the behavior of Alternating Direction Method of Multipliers (ADMM). The Generalized Linear Model (GLM) algorithm uses these settings.

Table 62-36 DBMS_DATA_MINING ADMM Settings

Settings Name	Setting Value	Description
`ADMM_CONSENSUS`	A positive integer	It is a ADMM's consensus parameter. The value must be a positive number. The default value is `0.1`.

Table 62-36 (Cont.) DBMS_DATA_MINING ADMM Settings

Settings Name	Setting Value	Description
`ADMM_ITERATIONS`	A positive integer	The number of `ADMM` iterations. The value must be a positive integer. The default value is `50`.
`ADMM_TOLERANCE`	A positive integer	It is a tolerance parameter. The value must be a positive number. The default value is `0.0001`

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model
  The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.
- *Oracle Machine Learning for SQL Concepts*

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about neural network

DBMS_DATA_MINING — Solver Settings: LBFGS

The settings listed in the following table configure the behavior of L-BFGS. Neural Network and Generalized Linear Model (GLM) use these settings.

Table 62-37 DBMS_DATA_MINING L-BFGS Settings

Setting Name	Setting Value	Description
`LBFGS_GRADIENT_TOLERANCE`	An integer greater than 0 represented as a character string	Defines gradient infinity norm tolerance for L-BFGS. Default value is `1E-9`. Expression: `TO_CHAR (0.000000002)`
`LBFGS_HISTORY_DEPTH`	A positive integer.	Defines the number of historical copies kept in L-BFGS solver. The default value is `20`.
`LBFGS_SCALE_HESSIAN`	`LBFGS_SCALE_HESSIAN_ENABLE` `LBFGS_SCALE_HESSIAN_DISABLE`	Defines whether to scale Hessian in L-BFGS or not. Default value is `LBFGS_SCALE_HESSIAN_ENABLE`.

Related Topics

- DBMS_DATA_MINING — Algorithm Settings: Neural Network
  The settings listed in the following table configure the behavior of the Neural Network algorithm.

- DBMS_DATA_MINING — Algorithm Settings: Generalized Linear Model
  The settings listed in the following table configure the behavior of the Generalized Linear Model algorithm.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about neural network

DBMS_DATA_MINING Datatypes

The `DBMS_DATA_MINING` package defines object data types for processing transactional data. The package also defines a type for user-specified transformations. These types are called `DM_NESTED_`*n*, where *n* identifies the Oracle data type of the nested attributes.

The Oracle Machine Learning for SQL object data types are described in the following table:

Table 62-38 DBMS_DATA_MINING Summary of Data Types

Datatype	Description
`DM_NESTED_BINARY_DOUBLE`	The name and value of a numerical attribute of type `BINARY_DOUBLE`.
`DM_NESTED_BINARY_DOUBLES`	A collection of `DM_NESTED_BINARY_DOUBLE`.
`DM_NESTED_BINARY_FLOAT`	The name and value of a numerical attribute of type `BINARY_FLOAT`.
`DM_NESTED_BINARY_FLOATS`	A collection of `DM_NESTED_BINARY_FLOAT`.
`DM_NESTED_CATEGORICAL`	The name and value of a categorical attribute of type `CHAR`, `VARCHAR`, or `VARCHAR2`.
`DM_NESTED_CATEGORICALS`	A collection of `DM_NESTED_CATEGORICAL`.
`DM_NESTED_NUMERICAL`	The name and value of a numerical attribute of type `NUMBER` or `FLOAT`.
`DM_NESTED_NUMERICALS`	A collection of `DM_NESTED_NUMERICAL`.
`ORA_MINING_VARCHAR2_NT`	A table of `VARCHAR2(4000)`.
`TRANSFORM_LIST`	A list of user-specified transformations for a model. Accepted as a parameter by the CREATE_MODEL Procedure. This collection type is defined in the DBMS_DATA_MINING_TRANSFORM package.

For more information about processing nested data, see *Oracle Machine Learning for SQL User's Guide*.

> **Note:**
>
> Starting from Oracle Database 12*c* Release 2, `*GET_MODEL_DETAILS` are deprecated and are replaced with *Model Detail Views*. See *Oracle Machine Learning for SQL User's Guide*.

Deprecated Types

This topic contains tables listing deprecated types.

The `DBMS_DATA_MINING` package defines object datatypes for storing information about model attributes. Most of these types are returned by the table functions `GET_n`, where *n* identifies the type of information to return. These functions take a model name as input and return the requested information as a collection of rows.

For a list of the `GET` functions, see "Summary of DBMS_DATA_MINING Subprograms".

All the table functions use pipelining, which causes each row of output to be materialized as it is read from model storage, without waiting for the generation of the complete table object. For more information on pipelined, parallel table functions, consult the *Oracle Database PL/SQL Language Reference.*

Table 62-39 DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
`DM_CENTROID`	The centroid of a cluster.
`DM_CENTROIDS`	A collection of `DM_CENTROID`. A member of `DM_CLUSTER`.
`DM_CHILD`	A child node of a cluster.
`DM_CHILDREN`	A collection of `DM_CHILD`. A member of `DM_CLUSTER`.
`DM_CLUSTER`	A cluster. A cluster includes `DM_PREDICATES`, `DM_CHILDREN`, `DM_CENTROIDS`, and `DM_HISTOGRAMS`. It also includes a `DM_RULE`. See also, DM_CLUSTER Fields.
`DM_CLUSTERS`	A collection of `DM_CLUSTER`. Returned by GET_MODEL_DETAILS_KM Function, GET_MODEL_DETAILS_OC Function, and GET_MODEL_DETAILS_EM Function. See also, DM_CLUSTER Fields.
`DM_CONDITIONAL`	The conditional probability of an attribute in a Naive Bayes model.
`DM_CONDITIONALS`	A collection of `DM_CONDITIONAL`. Returned by GET_MODEL_DETAILS_NB Function.
`DM_COST_ELEMENT`	The actual and predicted values in a cost matrix.
`DM_COST_MATRIX`	A collection of `DM_COST_ELEMENT`. Returned by GET_MODEL_COST_MATRIX Function.
`DM_EM_COMPONENT`	A component of an Expectation Maximization model.
`DM_EM_COMPONENT_SET`	A collection of `DM_EM_COMPONENT`. Returned by GET_MODEL_DETAILS_EM_COMP Function.
`DM_EM_PROJECTION`	A projection of an Expectation Maximization model.
`DM_EM_PROJECTION_SET`	A collection of `DM_EM_PROJECTION`. Returned by GET_MODEL_DETAILS_EM_PROJ Function.
`DM_GLM_COEFF`	The coefficient and associated statistics of an attribute in a Generalized Linear Model.
`DM_GLM_COEFF_SET`	A collection of `DM_GLM_COEFF`. Returned by GET_MODEL_DETAILS_GLM Function.
`DM_HISTOGRAM_BIN`	A histogram associated with a cluster.

Table 62-39 (Cont.) DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
DM_HISTOGRAMS	A collection of DM_HISTOGRAM_BIN. A member of DM_CLUSTER. See also, DM_CLUSTER Fields.
DM_ITEM	An item in an association rule.
DM_ITEMS	A collection of DM_ITEM.
DM_ITEMSET	A collection of DM_ITEMS.
DM_ITEMSETS	A collection of DM_ITEMSET. Returned by GET_FREQUENT_ITEMSETS Function.
DM_MODEL_GLOBAL_DETAIL	High-level statistics about a model.
DM_MODEL_GLOBAL_DETAILS	A collection of DM_MODEL_GLOBAL_DETAIL. Returned by GET_MODEL_DETAILS_GLOBAL Function.
DM_NB_DETAIL	Information about an attribute in a Naive Bayes model.
DM_NB_DETAILS	A collection of DM_DB_DETAIL. Returned by GET_MODEL_DETAILS_NB Function.
DM_NMF_ATTRIBUTE	An attribute in a feature of a Non-Negative Matrix Factorization model.
DM_NMF_ATTRIBUTE_SET	A collection of DM_NMF_ATTRIBUTE. A member of DM_NMF_FEATURE.
DM_NMF_FEATURE	A feature in a Non-Negative Matrix Factorization model.
DM_NMF_FEATURE_SET	A collection of DM_NMF_FEATURE. Returned by GET_MODEL_DETAILS_NMF Function.
DM_PREDICATE	Antecedent and consequent in a rule.
DM_PREDICATES	A collection of DM_PREDICATE. A member of DM_RULE and DM_CLUSTER. Predicates are returned by GET_ASSOCIATION_RULES Function, GET_MODEL_DETAILS_EM Function, GET_MODEL_DETAILS_KM Function, and GET_MODEL_DETAILS_OC Function. See also, DM_CLUSTER Fields.
DM_RANKED_ATTRIBUTE	An attribute ranked by its importance in an Attribute Importance model.
DM_RANKED_ATTRIBUTES	A collection of DM_RANKED_ATTRIBUTE. Returned by GET_MODEL_DETAILS_AI Function.
DM_RULE	A rule that defines a conditional relationship. The rule can be one of the association rules returned by GET_ASSOCIATION_RULES Function, or it can be a rule associated with a cluster in the collection of clusters returned by GET_MODEL_DETAILS_KM Function and GET_MODEL_DETAILS_OC Function. See also, DM_CLUSTER Fields.
DM_RULES	A collection of DM_RULE. Returned by GET_ASSOCIATION_RULES Function. See also, DM_CLUSTER Fields.
DM_SVD_MATRIX	A factorized matrix S, V, or U returned by a Singular Value Decomposition model.

Table 62-39 (Cont.) DBMS_DATA_MINING Summary of Deprecated Datatypes

Datatype	Description
`DM_SVD_MATRIX_SET`	A collection of `DM_SVD_MATRIX`. Returned by GET_MODEL_DETAILS_SVD Function.
`DM_SVM_ATTRIBUTE`	The name, value, and coefficient of an attribute in a Support Vector Machine model.
`DM_SVM_ATTRIBUTE_SET`	A collection of `DM_SVM_ATTRIBUTE`. Returned by GET_MODEL_DETAILS_SVM Function. Also a member of `DM_SVM_LINEAR_COEFF`.
`DM_SVM_LINEAR_COEFF`	The linear coefficient of each attribute in a Support Vector Machine model.
`DM_SVM_LINEAR_COEFF_SET`	A collection of `DM_SVM_LINEAR_COEFF`. Returned by GET_MODEL_DETAILS_SVM Function for an SVM model built using the linear kernel.
`DM_TRANSFORM`	The transformation and reverse transformation expressions for an attribute.
`DM_TRANSFORMS`	A collection of `DM_TRANSFORM`. Returned by GET_MODEL_TRANSFORMATIONS Function.

Return Values for Clustering Algorithms

The table contains description of `DM_CLUSTER` return value columns, nested table columns, and rows.

Table 62-40 DM_CLUSTER Return Values for Clustering Algorithms

<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td><code>DM_CLUSTERS</code></td><td>A set of rows of type <code>DM_CLUSTER</code>. The rows have the following columns:
<pre>
(id   NUMBER,
 cluster_id   VARCHAR2(4000),
 record_count   NUMBER,
 parent   NUMBER,
 tree_level   NUMBER,
 dispersion   NUMBER,
 split_predicate   DM_PREDICATES,
 child   DM_CHILDREN,
 centroid   DM_CENTROIDS,
 histogram   DM_HISTOGRAMS,
 rule   DM_RULE)
</pre></td></tr>
<tr><td><code>DM_PREDICATE</code></td><td>The <code>antecedent</code> and <code>consequent</code> columns each return nested tables of type <code>DM_PREDICATES</code>. The rows, of type <code>DM_PREDICATE</code>, have the following columns:
<pre>
(attribute_name   VARCHAR2(4000),
 attribute_subname   VARCHAR2(4000),
 conditional_operator   CHAR(2)/*=,<>,<,>,<=,>=*/,
 attribute_num_value   NUMBER,
 attribute_str_value   VARCHAR2(4000),
 attribute_support   NUMBER,
 attribute_confidence   NUMBER)
</pre></td></tr>
</table>

DM_CLUSTER Fields

The following table describes DM_CLUSTER fields.

Table 62-41 DM_CLUSTER Fields

Column Name	Description
id	Cluster identifier
cluster_id	The ID of a cluster in the model
record_count	Specifies the number of records
parent	Parent ID
tree_level	Specifies the number of splits from the root
dispersion	A measure used to quantify whether a set of observed occurrences are dispersed compared to a standard statistical model.

split_predicate	The split_predicate column of DM_CLUSTER returns a nested table of type DM_PREDICATES. Each row, of type DM_PREDICATE, has the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`conditional_operator CHAR(2) /`
	`*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`
	
	Note: The Expectation Maximization algorithm uses all the fields except dispersion and split_predicate.

child	The child column of DM_CLUSTER returns a nested table of type DM_CHILDREN. The rows, of type DM_CHILD, have a single column of type NUMBER, which contains the identifiers of each child.

centroid	The centroid column of DM_CLUSTER returns a nested table of type DM_CENTROIDS. The rows, of type DM_CENTROID, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`mean NUMBER,`
	`mode_value VARCHAR2(4000),`
	`variance NUMBER)`

Table 62-41 (Cont.) DM_CLUSTER Fields

Column Name	Description

`histogram`	The `histogram` column of `DM_CLUSTER` returns a nested table of type `DM_HISTOGRAMS`. The rows, of type `DM_HISTOGRAM_BIN`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`bin_id NUMBER,`
	`lower_bound NUMBER,`
	`upper_bound NUMBER,`
	`label VARCHAR2(4000),`
	`count NUMBER)`

`rule`	The `rule` column of `DM_CLUSTER` returns a single row of type `DM_RULE`. The columns are:
	`(rule_id INTEGER,`
	`antecedent DM_PREDICATES,`
	`consequent DM_PREDICATES,`
	`rule_support NUMBER,`
	`rule_confidence NUMBER,`
	`rule_lift NUMBER,`
	`antecedent_support NUMBER,`
	`consequent_support NUMBER,`
	`number_of_items INTEGER)`

Usage Notes

- The table function pipes out rows of type `DM_CLUSTER`. For information on Oracle Machine Learning for SQL data types and piped output from table functions, see "Data Types".
- For descriptions of predicates (`DM_PREDICATE`) and rules (`DM_RULE`), see GET_ASSOCIATION_RULES Function.

Summary of DBMS_DATA_MINING Subprograms

This table summarizes the subprograms included in the `DBMS_DATA_MINING` package.

The `GET_*` interfaces are replaced by model views. Oracle recommends that users leverage model detail views instead. For more information, refer to Model Detail Views in *Oracle Machine Learning for SQL User's Guide* and Static Data Dictionary Views: ALL_ALL_TABLES to ALL_OUTLINES in *Oracle Database Reference*.

Table 62-42 DBMS_DATA_MINING Package Subprograms

Subprogram	Purpose
ADD_COST_MATRIX Procedure	Adds a cost matrix to a classification model
ADD_PARTITION Procedure	Adds single or multiple partitions in an existing partition model
ALTER_REVERSE_EXPRESSION Procedure	Changes the reverse transformation expression to an expression that you specify
APPLY Procedure	Applies a model to a data set (scores the data)

Table 62-42 (Cont.) DBMS_DATA_MINING Package Subprograms

Subprogram	Purpose
COMPUTE_CONFUSION_MATRIX Procedure	Computes the confusion matrix for a classification model
COMPUTE_CONFUSION_MATRIX_PART Procedure	Computes the evaluation matrix for partitioned models
COMPUTE_LIFT Procedure	Computes lift for a classification model
COMPUTE_LIFT_PART Procedure	Computers lift for partitioned models
COMPUTE_ROC Procedure	Computes Receiver Operating Characteristic (ROC) for a classification model
COMPUTE_ROC_PART Procedure	Computes Receiver Operating Characteristic (ROC) for a partitioned model
CREATE_MODEL Procedure	Creates a model
CREATE_MODEL2 Procedure	Creates a model without extra persistent stages
Create Model Using Registration Information	Fetches setting information from JSON object
DROP_ALGORITHM Procedure	Drops the registered algorithm information.
DROP_PARTITION Procedure	Drops a single partition
DROP_MODEL Procedure	Drops a model
EXPORT_MODEL Procedure	Exports a model to a dump file
EXPORT_SERMODEL Procedure	Exports a model in a serialized format
FETCH_JSON_SCHEMA Procedure	Fetches and reads JSON schema from `all_mining_algorithms` view
GET_MODEL_COST_MATRIX Function	Returns the cost matrix for a model
IMPORT_MODEL Procedure	Imports a model into a user schema
IMPORT_ONNX_MODEL Procedure	Imports an ONNX model into the Database
IMPORT_SERMODEL Procedure	Imports a serialized model back into the database
JSON Schema for R Extensible Algorithm	Displays flexibility in creating JSON schema for R Extensible
REGISTER_ALGORITHM Procedure	Registers a new algorithm
RANK_APPLY Procedure	Ranks the predictions from the `APPLY` results for a classification model
REMOVE_COST_MATRIX Procedure	Removes a cost matrix from a model
RENAME_MODEL Procedure	Renames a model

Deprecated GET_MODEL_DETAILS

Starting from Oracle Database 12*c* Release 2, the following `GET_MODEL_DETAILS` are deprecated:

Table 62-43 Deprecated `GET_MODEL_DETAILS` Functions

Subprogram	Purpose
GET_ASSOCIATION_RULES Function	Returns the rules from an association model

Table 62-43 (Cont.) Deprecated `GET_MODEL_DETAILS` Functions

Subprogram	Purpose
GET_FREQUENT_ITEMSETS Function	Returns the frequent itemsets for an association model
GET_MODEL_DETAILS_AI Function	Returns details about an attribute importance model
GET_MODEL_DETAILS_EM Function	Returns details about an Expectation Maximization model
GET_MODEL_DETAILS_EM_COMP Function	Returns details about the parameters of an Expectation Maximization model
GET_MODEL_DETAILS_EM_PROJ Function	Returns details about the projects of an Expectation Maximization model
GET_MODEL_DETAILS_GLM Function	Returns details about a Generalized Linear Model model
GET_MODEL_DETAILS_GLOBAL Function	Returns high-level statistics about a model
GET_MODEL_DETAILS_KM Function	Returns details about a *k*-Means model
GET_MODEL_DETAILS_NB Function	Returns details about a Naive Bayes model
GET_MODEL_DETAILS_NMF Function	Returns details about a Non-Negative Matrix Factorization model
GET_MODEL_DETAILS_OC Function	Returns details about an O-Cluster model
GET_MODEL_SETTINGS Function	Returns the settings used to build the given model This function is replaced with `USER/ALL/DBA_MINING_MODEL_SETTINGS`
GET_MODEL_SIGNATURE Function	Returns the list of columns from the build input table This function is replaced with `USER/ALL/DBA_MINING_MODEL_ATTRIBUTES`
GET_MODEL_DETAILS_SVD Function	Returns details about a Singular Value Decomposition model
GET_MODEL_DETAILS_SVM Function	Returns details about a Support Vector Machine model with a linear kernel
GET_MODEL_TRANSFORMATIONS Function	Returns the transformations embedded in a model This function is replaced with `USER/ALL/DBA_MINING_MODEL_XFORMS`
GET_MODEL_DETAILS_XML Function	Returns details about a Decision Tree model
GET_TRANSFORM_LIST Procedure	Converts between two different transformation specification formats

Related Topics

- *Oracle Machine Learning for SQL User's Guide*
- *Oracle Database Reference*

ADD_COST_MATRIX Procedure

The `ADD_COST_MATRIX` procedure associates a cost matrix table with a classification model. The cost matrix biases the model by assigning costs or benefits to specific model outcomes.

The cost matrix is stored with the model and taken into account when the model is scored.

You can also specify a cost matrix inline when you invoke an Oracle Machine Learning for SQL function for scoring. To view the scoring matrix for a model, query the `DM$VC` prefixed model view. Refer to Model Detail View for Classification Algorithm.

To obtain the default scoring matrix for a model, query the `DM$VC` prefixed model view. To remove the default scoring matrix from a model, use the `REMOVE_COST_MATRIX` procedure. See REMOVE_COST_MATRIX Procedure.

See Also:

- "Biasing a Classification Model" in *Oracle Machine Learning for SQL Concepts* for more information about costs
- *Oracle Database SQL Language Reference* for syntax of inline cost matrix
- Specifying Costs in *Oracle Machine Learning for SQL User's Guide*

Syntax

```
DBMS_DATA_MINING.ADD_COST_MATRIX (
       model_name                IN VARCHAR2,
       cost_matrix_table_name    IN VARCHAR2,
       cost_matrix_schema_name   IN VARCHAR2 DEFAULT NULL);
       partition_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-44 ADD_COST_MATRIX Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is assumed.
`cost_matrix_table_name`	Name of the cost matrix table (described in Table 62-45).
`cost_matrix_schema_name`	Schema of the cost matrix table. If no schema is specified, then the current schema is used.
`partition_name`	Name of the partition in a partitioned model

Usage Notes

1. If the model is not in your schema, then `ADD_COST_MATRIX` requires the `ALTER ANY MINING MODEL` system privilege or the `ALTER` object privilege for the machine learning model.
2. The cost matrix table must have the columns shown in Table 62-45.

Table 62-45 Required Columns in a Cost Matrix Table

Column Name	Data Type
ACTUAL_TARGET_VALUE	Valid target data type
PREDICTED_TARGET_VALUE	Valid target data type
COST	NUMBER, FLOAT, BINARY_DOUBLE, or BINARY_FLOAT

See Also:

Oracle Machine Learning for SQL User's Guide for valid target data types

3. The types of the actual and predicted target values must be the same as the type of the model target. For example, if the target of the model is BINARY_DOUBLE, then the actual and predicted values must be BINARY_DOUBLE. If the actual and predicted values are CHAR or VARCHAR, then ADD_COST_MATRIX treats them as VARCHAR2 internally.

   If the types do not match, or if the actual or predicted value is not a valid target value, then the ADD_COST_MATRIX procedure raises an error.

   Note:

   If a reverse transformation is associated with the target, then the actual and predicted values must be consistent with the target after the reverse transformation has been applied.

   See "Reverse Transformations and Model Transparency" under the "About Transformation Lists" section in DBMS_DATA_MINING_TRANSFORM Operational Notes for more information.

4. Since a benefit can be viewed as a negative cost, you can specify a benefit for a given outcome by providing a negative number in the costs column of the cost matrix table.
5. All classification algorithms can use a cost matrix for scoring. The Decision Tree algorithm can also use a cost matrix at build time. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the CLAS_COST_TABLE_NAME setting in the settings table for the model. See Table 62-7.

   The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, use the REMOVE_COST_MATRIX procedure to remove the cost matrix and the ADD_COST_MATRIX procedure to add a new one.
6. Scoring on a partitioned model is partition-specific. Scoring cost matrices can be added to or removed from an individual partition in a partitioned model. If PARTITION_NAME is NOT NULL, then the model must be a partitioned model. The COST_MATRIX is added to that partition of the partitioned model.

   If the PARTITION_NAME is NULL, but the model is a partitioned model, then the COST_MATRIX table is added to every partition in the model.

Example

This example creates a cost matrix table called `COSTS_NB` and adds it to a Naive Bayes model called `NB_SH_CLAS_SAMPLE`. The model has a binary target: 1 means that the customer responds to a promotion; 0 means that the customer does not respond. The cost matrix assigns a cost of .25 to misclassifications of customers who do not respond and a cost of .75 to misclassifications of customers who do respond. This means that it is three times more costly to misclassify responders than it is to misclassify non-responders.

```
CREATE TABLE costs_nb (
  actual_target_value           NUMBER,
  predicted_target_value        NUMBER,
  cost                          NUMBER);
INSERT INTO costs_nb values (0, 0, 0);
INSERT INTO costs_nb values (0, 1, .25);
INSERT INTO costs_nb values (1, 0, .75);
INSERT INTO costs_nb values (1, 1, 0);
COMMIT;

EXEC dbms_data_mining.add_cost_matrix('nb_sh_clas_sample', 'costs_nb');

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;

C        CNT    AVG_AGE
- ---------- ----------
F         72         39
M        555         44
```

ADD_PARTITION Procedure

`ADD_PARTITION` procedure supports a single or multiple partition addition to an existing partitioned model.

The `ADD_PARTITION` procedure derives build settings and user-defined expressions from the existing model. The target column must exist in the input data query when adding partitions to a supervised model.

Syntax

```
DBMS_DATA_MINING.ADD_PARTITION (
      model_name                 IN VARCHAR2,
      data_query                 IN CLOB,
      add_options                IN VARCHAR2 DEFAULT ERROR);
```

Parameters

Table 62-46 ADD_PARTITION Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Table 62-46 (Cont.) ADD_PARTITION Procedure Parameters

Parameter	Description
`data_query`	An arbitrary SQL statement that provides data to the model build. The user must have privilege to evaluate this query.
`add_options`	Allows users to control the conditional behavior of ADD for cases where rows in the input dataset conflict with existing partitions in the model. The following are the possible values: • `REPLACE`: Replaces the existing partition for which the conflicting keys are found. • `ERROR`: Terminates the ADD operation without adding any partitions. • `IGNORE`: Eliminates the rows having the conflicting keys. **Note:** For better performance, Oracle recommends using `DROP_PARTITION` followed by the `ADD_PARTITION` instead of using the `REPLACE` option.

ALTER_REVERSE_EXPRESSION Procedure

This procedure replaces a reverse transformation expression with an expression that you specify. If the attribute does not have a reverse expression, the procedure creates one from the specified expression.

You can also use this procedure to customize the output of clustering, feature extraction, and anomaly detection models.

Syntax

```
DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION (
         model_name              VARCHAR2,
         expression              CLOB,
         attribute_name          VARCHAR2 DEFAULT NULL,
         attribute_subname       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-47 ALTER_REVERSE_EXPRESSION Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used.
`expression`	An expression to replace the reverse transformation associated with the attribute.
`attribute_name`	Name of the attribute. Specify `NULL` if you wish to apply *`expression`* to a cluster, feature, or One-Class SVM prediction.
`attribute_subname`	Name of the nested attribute if *`attribute_name`* is a nested column, otherwise `NULL`.

Usage Notes

1. For purposes of model transparency, Oracle Machine Learning for SQL provides reverse transformations for transformations that are embedded in a model. Reverse transformations are applied to the attributes returned in model detail views and to the scored target of predictive models.

   > **See Also:**
   >
   > - “About Transformation Lists” under DBMS_DATA_MINING_TRANSFORM Operational Notes
   > - Model Detail Views in *Oracle Machine Learning for SQL User’s Guide*

2. If you alter the reverse transformation for the target of a model that has a cost matrix, you must specify a transformation expression that has the same type as the actual and predicted values in the cost matrix. Also, the reverse transformation that you specify must result in values that are present in the cost matrix.

   > **See Also:**
   >
   > "ADD_COST_MATRIX Procedure" and *Oracle Machine Learning for SQL Concepts* for information about cost matrixes.

3. To prevent reverse transformation of an attribute, you can specify `NULL` for *`expression`*.

4. The reverse transformation expression can contain a reference to a PL/SQL function that returns a valid Oracle data type. For example, you could define a function like the following for a categorical attribute named `blood_pressure` that has values 'Low', 'Medium' and 'High'.

```
CREATE OR REPLACE FUNCTION numx(c char) RETURN NUMBER IS
  BEGIN
    CASE c WHEN ''Low'' THEN RETURN 1;
           WHEN ''Medium'' THEN RETURN 2;
           WHEN ''High'' THEN RETURN 3;
           ELSE RETURN null;
    END CASE;
  END numx;
```

   Then you could invoke `ALTER_REVERSE_EXPRESION` for `blood_pressure` as follows.

```
EXEC dbms_data_mining.alter_reverse_expression(
             '<model_name>', 'NUMX(blood_pressure)', 'blood_pressure');
```

5. You can use `ALTER_REVERSE_EXPRESSION` to label clusters produced by clustering models and features produced by feature extraction.

   You can use `ALTER_REVERSE_EXPRESSION` to replace the zeros and ones returned by anomaly-detection models. By default, anomaly-detection models label anomalous records with 0 and all other records with 1.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for information about anomaly detection

Examples

1. In this example, the target (`affinity_card`) of the model `CLASS_MODEL` is manipulated internally as `yes` or `no` instead of `1` or `0` but returned as `1`s and `0`s when scored. The `ALTER_REVERSE_EXPRESSION` procedure causes the target values to be returned as `TRUE` or `FALSE`.

```
DECLARE
        v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
          'affinity_card', NULL,
          'decode(affinity_card, 1, ''yes'', ''no'')',
          'decode(affinity_card, ''yes'', 1, 0)');
    dbms_data_mining.CREATE_MODEL(
      model_name             => 'CLASS_MODEL',
      mining_function        => dbms_data_mining.classification,
      data_table_name        => 'mining_data_build',
      case_id_column_name    => 'cust_id',
      target_column_name     => 'affinity_card',
      settings_table_name    => NULL,
      data_schema_name       => 'oml_user',
      settings_schema_name   => NULL,
      xform_list             => v_xlst );
  END;
/
SELECT cust_income_level, occupation,
           PREDICTION(CLASS_MODEL USING *) predict_response
      FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
      ORDER BY cust_income_level;

CUST_INCOME_LEVEL              OCCUPATION            PREDICT_RESPONSE
------------------------------ --------------------- --------------------
A: Below 30,000                Transp.                                  1
E: 90,000 - 109,999            Transp.                                  1
E: 90,000 - 109,999            Sales                                    1
G: 130,000 - 149,999           Handler                                  0
G: 130,000 - 149,999           Crafts                                   0
H: 150,000 - 169,999           Prof.                                    1
J: 190,000 - 249,999           Prof.                                    1
J: 190,000 - 249,999           Sales                                    1

BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION (
     model_name      => 'CLASS_MODEL',
     expression      => 'decode(affinity_card, ''yes'', ''TRUE'', ''FALSE'')',
     attribute_name  => 'affinity_card');
END;
/
column predict_response on
column predict_response format a20
SELECT cust_income_level, occupation,
             PREDICTION(CLASS_MODEL USING *) predict_response
      FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
```

```
      ORDER BY cust_income_level;

CUST_INCOME_LEVEL              OCCUPATION            PREDICT_RESPONSE
------------------------------ --------------------- --------------------
A: Below 30,000                Transp.               TRUE
E: 90,000 - 109,999            Transp.               TRUE
E: 90,000 - 109,999            Sales                 TRUE
G: 130,000 - 149,999           Handler               FALSE
G: 130,000 - 149,999           Crafts                FALSE
H: 150,000 - 169,999           Prof.                 TRUE
J: 190,000 - 249,999           Prof.                 TRUE
J: 190,000 - 249,999           Sales                 TRUE
```

2. This example specifies labels for the clusters that result from the `sh_clus` model. The labels consist of the word "Cluster" and the internal numeric identifier for the cluster.

```
BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION( 'sh_clus', '''Cluster ''||value');
END;
/

SELECT cust_id, cluster_id(sh_clus using *) cluster_id
   FROM sh_aprep_num
       WHERE cust_id < 100011
       ORDER by cust_id;

CUST_ID CLUSTER_ID
------- ------------------------------------------------
 100001 Cluster 18
 100002 Cluster 14
 100003 Cluster 14
 100004 Cluster 18
 100005 Cluster 19
 100006 Cluster 7
 100007 Cluster 18
 100008 Cluster 14
 100009 Cluster 8
 100010 Cluster 8
```

APPLY Procedure

The `APPLY` procedure applies a machine learning model to the data of interest, and generates the results in a table. The `APPLY` procedure is also referred to as **scoring**.

For predictive machine learning functions, the `APPLY` procedure generates predictions in a target column. For descriptive machine learning functions such as Clustering, the `APPLY` process assigns each case to a cluster with a probability.

In Oracle Machine Learning for SQL, the `APPLY` procedure is not applicable to Association models and Attribute Importance models.

Note:

Scoring can also be performed directly in SQL using the OML4SQL functions. See

- Oracle Machine Learning for SQL Functions in *Oracle Database SQL Language Reference*
- Scoring and Deployment in *Oracle Machine Learning for SQL User’s Guide*

Syntax

```
DBMS_DATA_MINING.APPLY (
      model_name           IN VARCHAR2,
      data_table_name      IN VARCHAR2,
      case_id_column_name  IN VARCHAR2,
      result_table_name    IN VARCHAR2,
      data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-48 ***APPLY Procedure Parameters***

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`data_table_name`	Name of table or view containing the data to be scored
`case_id_column_name`	Name of the case identifier column
`result_table_name`	Name of the table in which to store apply results
`data_schema_name`	Name of the schema containing the data to be scored

Usage Notes

1. The data provided for `APPLY` must undergo the same preprocessing as the data used to create and test the model. When you use Automatic Data Preparation, the preprocessing required by the algorithm is handled for you by the model: both at build time and apply time. (See "Automatic Data Preparation".)
2. `APPLY` creates a table in the user's schema to hold the results. The columns are algorithm-specific.

   The columns in the results table are listed in Table 62-49 through Table 62-53. The case ID column name in the results table will match the case ID column name provided by you. The type of the incoming case ID column is also preserved in `APPLY` output.

   > **Note:**
   >
   > Make sure that the case ID column does not have the same name as one of the columns that will be created by `APPLY`. For example, when applying a Classification model, the case ID in the scoring data must not be `PREDICTION` or `PROBABILITY` (See Table 62-49).

3. The data type for the `PREDICTION`, `CLUSTER_ID`, and `FEATURE_ID` output columns is influenced by any reverse expression that is embedded in the model by the user. If the user does not provide a reverse expression that alters the scored value type, then the types will conform to the descriptions in the following tables. See "ALTER_REVERSE_EXPRESSION Procedure".
4. If the model is partitioned, the `result_table_name` can contain results from different partitions depending on the data from the input data table. An additional column called `PARTITION_NAME` is added to the result table indicating the partition name that is associated with each row.

For a non-partitioned model, the behavior does not change.

Classification

The results table for Classification has the columns described in Table 62-49. If the target of the model is categorical, the `PREDICTION` column will have a `VARCHAR2` data type. If the target has a binary type, the `PREDICTION` column will have the binary type of the target.

Table 62-49 APPLY Results Table for Classification

Column Name	Data type
`Case ID column name`	Type of the case ID
`PREDICTION`	Type of the target
`PROBABILITY`	`BINARY_DOUBLE`

Anomaly Detection

The results table for Anomaly Detection has the columns described in Table 62-50.

Table 62-50 APPLY Results Table for Anomaly Detection

Column Name	Data Type
`Case ID column name`	Type of the case ID
`PREDICTION`	`NUMBER`
`PROBABILITY`	`BINARY_DOUBLE`

Regression

The results table for Regression has the columns described in APPLY Procedure.

Table 62-51 APPLY Results Table for Regression

Column Name	Data Type
`Case ID column name`	Type of the case ID
`PREDICTION`	Type of the target

Clustering

Clustering is an unsupervised machine learning function, and hence there are no targets. The results of an `APPLY` procedure contain simply the cluster identifier corresponding to a case, and the associated probability. The results table has the columns described in Table 62-52.

Table 62-52 APPLY Results Table for Clustering

Column Name	Data Type
`Case ID column name`	Type of the case ID
`CLUSTER_ID`	`NUMBER`
`PROBABILITY`	`BINARY_DOUBLE`

Feature Extraction

Feature Extraction is also an unsupervised machine learning function, hence there are no targets. The results of an `APPLY` procedure will contain simply the feature identifier corresponding to a case, and the associated match quality. The results table has the columns described in Table 62-53.

Table 62-53 APPLY Results Table for Feature Extraction

Column Name	Data Type
`Case ID column name`	Type of the case ID
`FEATURE_ID`	`NUMBER`
`MATCH_QUALITY`	`BINARY_DOUBLE`

Examples

This example applies the GLM Regression model `GLMR_SH_REGR_SAMPLE` to the data in the `MINING_DATA_APPLY_V` view. The `APPLY` results are output of the table `REGRESSION_APPLY_RESULT`.

```
SQL> BEGIN
       DBMS_DATA_MINING.APPLY (
       model_name     => 'glmr_sh_regr_sample',
       data_table_name     => 'mining_data_apply_v',
       case_id_column_name => 'cust_id',
       result_table_name   => 'regression_apply_result');
    END;
    /

SQL> SELECT * FROM regression_apply_result WHERE cust_id >  101485;

   CUST_ID PREDICTION
---------- ----------
    101486 22.8048824
    101487 25.0261101
    101488 48.6146619
    101489   51.82595
    101490 22.6220714
    101491 61.3856816
    101492 24.1400748
    101493  58.034631
    101494 45.7253149
    101495 26.9763318
    101496 48.1433425
    101497 32.0573434
    101498 49.8965531
    101499  56.270656
    101500 21.1153047
```

COMPUTE_CONFUSION_MATRIX Procedure

This procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

`COMPUTE_CONFUSION_MATRIX` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about confusion matrixes and other test metrics for classification
>
> "COMPUTE_LIFT Procedure"
>
> "COMPUTE_ROC Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
      accuracy                     OUT NUMBER,
      apply_result_table_name      IN  VARCHAR2,
      target_table_name            IN  VARCHAR2,
      case_id_column_name          IN  VARCHAR2,
      target_column_name           IN  VARCHAR2,
      confusion_matrix_table_name  IN  VARCHAR2,
      score_column_name            IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN  VARCHAR2 DEFAULT 'PROBABILITY',
      cost_matrix_table_name       IN  VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN  VARCHAR2 DEFAULT NULL,
      target_schema_name           IN  VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN  VARCHAR2 DEFAULT NULL,
      score_criterion_type         IN  VARCHAR2 DEFAULT 'PROBABILITY');
```

Parameters

Table 62-54 COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
`accuracy`	Output parameter containing the overall percentage accuracy of the predictions.
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.

Table 62-54 (Cont.) COMPUTE_CONFUSION_MATRIX Procedure Parameters

Parameter	Description
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`confusion_matrix_table_name`	Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is '`PROBABILITY'`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `'COSTS'`, the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table. If null, the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `'PROBABILITY'`. To use costs as the scoring criterion, specify `'COST'`. If `score_criterion_type` is set to '`COST`' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_CONFUSION_MATRIX` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to `COMPUTE_CONFUSION_MATRIX`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_CONFUSION_MATRIX` are in a table or view specified in `apply_result_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 62-55.

Table 62-55 Columns in a Cost Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the build data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`cost`	`BINARY_DOUBLE`

See Also:

Oracle Machine Learning for SQL User's Guide for valid target data types

Oracle Machine Learning for SQL Concepts for more information about cost matrixes

- The confusion matrix created by `COMPUTE_CONFUSION_MATRIX` has the columns described in Table 62-56.

Table 62-56 Columns in a Confusion Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the build data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`value`	`BINARY_DOUBLE`

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about confusion matrixes

Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`.

Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
       SELECT cust_id,
              PREDICTION(nb_sh_clas_sample USING *) prediction,
              PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
       FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```
DECLARE
   v_accuracy    NUMBER;
      BEGIN
        DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
                   accuracy                     => v_accuracy,
                   apply_result_table_name      => 'nb_apply_results',
                   target_table_name            => 'mining_data_test_v',
                   case_id_column_name          => 'cust_id',
                   target_column_name           => 'affinity_card',
                   confusion_matrix_table_name  => 'nb_confusion_matrix',
                   score_column_name            => 'PREDICTION',
                   score_criterion_column_name  => 'PROBABILITY'
                   cost_matrix_table_name       =>  null,
                   apply_result_schema_name     =>  null,
                   target_schema_name           =>  null,
                   cost_matrix_schema_name      =>  null,
                   score_criterion_type         => 'PROBABILITY');
        DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
      END;
      /
```

The confusion matrix and model accuracy are shown as follows.

```
 **** MODEL ACCURACY ****: .7847

SQL>SELECT * from nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         60
                  0                      0        891
                  1                      1        286
                  0                      1        263
```

Compute a Confusion Matrix Based on a Cost Matrix Table

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```
SQL> SELECT * from nb_cost_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1        .75
                  1                      0        .25
                  1                      1          0
```

This statement shows how to generate the predictions using `APPLY`.

```
BEGIN
    DBMS_DATA_MINING.APPLY(
          model_name          => 'nb_sh_clas_sample',
          data_table_name     => 'mining_data_test_v',
          case_id_column_name => 'cust_id',
          result_table_name   => 'nb_apply_results');
 END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named '`PROBABILITY`', which is the name generated by `APPLY`.

```
DECLARE
  v_accuracy    NUMBER;
     BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
                accuracy                     => v_accuracy,
                apply_result_table_name      => 'nb_apply_results',
                target_table_name            => 'mining_data_test_v',
                case_id_column_name          => 'cust_id',
                target_column_name           => 'affinity_card',
                confusion_matrix_table_name  => 'nb_confusion_matrix',
                score_column_name            => 'PREDICTION',
                score_criterion_column_name  => 'PROBABILITY',
                cost_matrix_table_name       => 'nb_cost_matrix',
                apply_result_schema_name     => null,
                target_schema_name           => null,
                cost_matrix_schema_name      => null,
                score_criterion_type         => 'COST');
       DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
    END;
    /
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```
**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

Compute a Confusion Matrix Based on Embedded Costs

You can use the `ADD_COST_MATRIX` procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN    DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample', 'nb_cost_matrix');END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
         SELECT cust_id,
              PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
              PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
          FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
   v_accuracy         NUMBER;
   BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
            accuracy                     => v_accuracy,
            apply_result_table_name      => 'nb_apply_results',
            target_table_name            => 'mining_data_test_v',
            case_id_column_name          => 'cust_id',
            target_column_name           => 'affinity_card',
            confusion_matrix_table_name  => 'nb_confusion_matrix',
            score_column_name            => 'PREDICTION',
            score_criterion_column_name  => 'COST',
            cost_matrix_table_name       => null,
            apply_result_schema_name     => null,
            target_schema_name           => null,
            cost_matrix_schema_name      => null,
            score_criterion_type         => 'COST');
   END;
   /
```

The results are:

```
**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

COMPUTE_CONFUSION_MATRIX_PART Procedure

The `COMPUTE_CONFUSION_MATRIX_PART` procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

`COMPUTE_CONFUSION_MATRIX_PART` provides support to computation of evaluation metrics per-partition for partitioned models. For non-partitioned models, refer to COMPUTE_CONFUSION_MATRIX Procedure.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

COMPUTE_CONFUSION_MATRIX_PART accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about confusion matrixes and other test metrics for classification
>
> "COMPUTE_LIFT_PART Procedure"
>
> "COMPUTE_ROC_PART Procedure"

Syntax

```
DBMS_DATA_MINING.compute_confusion_matrix_part(
      accuracy                    OUT DM_NESTED_NUMERICALS,
      apply_result_table_name     IN  VARCHAR2,
      target_table_name           IN  VARCHAR2,
      case_id_column_name         IN  VARCHAR2,
      target_column_name          IN  VARCHAR2,
      confusion_matrix_table_name IN  VARCHAR2,
      score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
      cost_matrix_table_name      IN  VARCHAR2 DEFAULT NULL,
      apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
      target_schema_name          IN  VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name     IN  VARCHAR2 DEFAULT NULL,
      score_criterion_type        IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-57 COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

Parameter	Description
accuracy	Output parameter containing the overall percentage accuracy of the predictions The output argument is changed from NUMBER to DM_NESTED_NUMERICALS

Table 62-57 (Cont.) COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

Parameter	Description
`apply_result_table_name`	Table containing the predictions
`target_table_name`	Table containing the known target values from the test data
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`confusion_matrix_table_name`	Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`score_partition_column_name`	(Optional) Parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `COSTS`, the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table. If null, then the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, then the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided. If null, then the user's schema is assumed.

Table 62-57 (Cont.) COMPUTE_CONFUSION_MATRIX_PART Procedure Parameters

Parameter	Description
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `PROBABILITY`. To use costs as the scoring criterion, specify `COST`. If `score_criterion_type` is set to `COST` but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_CONFUSION_MATRIX_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to `COMPUTE_CONFUSION_MATRIX_PART`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_CONFUSION_MATRIX_PART` are in a table or view specified in `apply_result_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name             VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 62-55.

Table 62-58 Columns in a Cost Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`cost`	`BINARY_DOUBLE`

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for valid target data types
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The confusion matrix created by `COMPUTE_CONFUSION_MATRIX_PART` has the columns described in Table 62-56.

Table 62-59 Columns in a Confusion Matrix Part

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target is the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`value`	`BINARY_DOUBLE`

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about confusion matrixes

Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`.

Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
       SELECT cust_id,
              PREDICTION(nb_sh_clas_sample USING *) prediction,
              PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
       FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```
DECLARE
   v_accuracy    NUMBER;
      BEGIN
        DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
                   accuracy                    => v_accuracy,
                   apply_result_table_name     => 'nb_apply_results',
                   target_table_name           => 'mining_data_test_v',
                   case_id_column_name         => 'cust_id',
                   target_column_name          => 'affinity_card',
                   confusion_matrix_table_name => 'nb_confusion_matrix',
                   score_column_name           => 'PREDICTION',
                   score_criterion_column_name => 'PROBABILITY'
                   score_partition_column_name => 'PARTITION_NAME'
```

```
                    cost_matrix_table_name       =>  null,
                    apply_result_schema_name     =>  null,
                    target_schema_name           =>  null,
                    cost_matrix_schema_name      =>  null,
                    score_criterion_type         => 'PROBABILITY');
      DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
    END;
    /
```

The confusion matrix and model accuracy are shown as follows.

```
 **** MODEL ACCURACY ****: .7847

SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         60
                  0                      0        891
                  1                      1        286
                  0                      1        263
```

Compute a Confusion Matrix Based on a Cost Matrix Table

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```
 SELECT * from NB_COST_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE       COST
------------------- ---------------------- ----------
                  0                      0          0
                  0                      1        .75
                  1                      0        .25
                  1                      1          0
```

This statement shows how to generate the predictions using `APPLY`.

```
BEGIN
    DBMS_DATA_MINING.APPLY(
          model_name          => 'nb_sh_clas_sample',
          data_table_name     => 'mining_data_test_v',
          case_id_column_name => 'cust_id',
          result_table_name   => 'nb_apply_results');
 END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named '`PROBABILITY`', which is the name generated by `APPLY`.

```
DECLARE
  v_accuracy    NUMBER;
     BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
                accuracy                     => v_accuracy,
                apply_result_table_name      => 'nb_apply_results',
                target_table_name            => 'mining_data_test_v',
                case_id_column_name          => 'cust_id',
                target_column_name           => 'affinity_card',
                confusion_matrix_table_name  => 'nb_confusion_matrix',
                score_column_name            => 'PREDICTION',
```

```
                score_criterion_column_name  => 'PROBABILITY',
                score_partition_column_name  => 'PARTITION_NAME'
                cost_matrix_table_name       => 'nb_cost_matrix',
                apply_result_schema_name     => null,
                target_schema_name           => null,
                cost_matrix_schema_name      => null,
                score_criterion_type         => 'COST');
       DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
    END;
    /
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```
**** MODEL ACCURACY ****: .798

 SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

Compute a Confusion Matrix Based on Embedded Costs

You can use the `ADD_COST_MATRIX` procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN
DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample', 'nb_cost_matrix');
END;/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
         SELECT cust_id,
              PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
              PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
          FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
   v_accuracy         NUMBER;
   BEGIN
       DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX_PART (
            accuracy                     => v_accuracy,
            apply_result_table_name      => 'nb_apply_results',
            target_table_name            => 'mining_data_test_v',
            case_id_column_name          => 'cust_id',
            target_column_name           => 'affinity_card',
            confusion_matrix_table_name  => 'nb_confusion_matrix',
            score_column_name            => 'PREDICTION',
            score_criterion_column_name  => 'COST',
            score_partition_column_name  => 'PARTITION_NAME'
            cost_matrix_table_name       => null,
            apply_result_schema_name     => null,
            target_schema_name           => null,
            cost_matrix_schema_name      => null,
            score_criterion_type         => 'COST');
```

```
END;
/
```

The results are:

```
**** MODEL ACCURACY ****: .798

 SELECT * FROM NB_CONFUSION_MATRIX;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
------------------- ---------------------- ----------
                  1                      0         91
                  0                      0        942
                  1                      1        255
                  0                      1        212
```

COMPUTE_LIFT Procedure

This procedure computes lift and stores the results in a table in the user's schema.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. `COMPUTE_LIFT` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

See Also:

Oracle Machine Learning for SQL Concepts for more details about lift and test metrics for classification

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_ROC Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT (
      apply_result_table_name      IN VARCHAR2,
      target_table_name            IN VARCHAR2,
      case_id_column_name          IN VARCHAR2,
      target_column_name           IN VARCHAR2,
      lift_table_name              IN VARCHAR2,
      positive_target_value        IN VARCHAR2,
      score_column_name            IN VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN VARCHAR2 DEFAULT 'PROBABILITY',
      num_quantiles                IN NUMBER DEFAULT 10,
      cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN VARCHAR2 DEFAULT NULL,
      target_schema_name           IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN VARCHAR2 DEFAULT NULL
      score_criterion_type         IN VARCHAR2 DEFAULT 'PROBABILITY');
```

Parameters

Table 62-60 COMPUTE_LIFT Procedure Parameters

Parameter	Description
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`lift_table_name`	Table containing the lift statistics. The table will be created by the procedure in the user's schema. The columns in the lift table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate lift. If the target column is a `NUMBER`, you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is '`PREDICTION`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is '`PROBABILITY`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.

Table 62-60 (Cont.) COMPUTE_LIFT Procedure Parameters

Parameter	Description
num_quantiles	Number of quantiles to be used in calculating lift. The default is 10.
cost_matrix_table_name	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to 'COST', the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
apply_result_schema_name	Schema of the apply results table. If null, the user's schema is assumed.
target_schema_name	Schema of the table containing the known targets. If null, the user's schema is assumed.
cost_matrix_schema_name	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.
score_criterion_type	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter. The default value of score_criterion_type is 'PROBABILITY'. To use costs as the scoring criterion, specify 'COST'. If score_criterion_type is set to 'COST' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to COMPUTE_LIFT may be generated using SQL PREDICTION functions, the DBMS_DATA_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the lift.
- Instead of passing a cost matrix to COMPUTE_LIFT, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION_COST function to populate the score criterion column.
- The predictions that you pass to COMPUTE_LIFT are in a table or view specified in apply_results_table_name.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 62-61.

Table 62-61 Columns in a Cost Matrix

Column Name	Data Type
actual_target_value	Type of the target column in the build data
predicted_target_value	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
cost	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The table created by COMPUTE_LIFT has the columns described in Table 62-62

Table 62-62 Columns in a Lift Table

Column Name	Data Type
quantile_number	NUMBER
probability_threshold	NUMBER
gain_cumulative	NUMBER
quantile_total_count	NUMBER
quantile_target_count	NUMBER
percent_records_cumulative	NUMBER
lift_cumulative	NUMBER
target_density_cumulative	NUMBER
targets_cumulative	NUMBER
non_targets_cumulative	NUMBER
lift_quantile	NUMBER
target_density	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for details about the information in the lift table

- When a cost matrix is passed to COMPUTE_LIFT, the cost threshold is returned in the probability_threshold column of the lift table.

Examples

This example uses the Naive Bayes model nb_sh_clas_sample.

The example illustrates lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute lift as follows.

```
BEGIN
       DBMS_DATA_MINING.COMPUTE_LIFT (
            apply_result_table_name            => 'nb_apply_results',
            target_table_name                  => 'mining_data_test_v',
            case_id_column_name                => 'cust_id',
            target_column_name                 => 'affinity_card',
            lift_table_name                        => 'nb_lift',
            positive_target_value              =>  to_char(1),
            score_column_name                  => 'PREDICTION',
            score_criterion_column_name    => 'PROBABILITY',
            num_quantiles                          =>  10,
            cost_matrix_table_name             =>  null,
            apply_result_schema_name           =>  null,
            target_schema_name                 =>  null,
            cost_matrix_schema_name            =>  null,
            score_criterion_type               =>  'PROBABILITY');
      END;
      /
```

This query displays some of the statistics from the resulting lift table.

```
SQL>SELECT quantile_number, probability_threshold, gain_cumulative,
           quantile_total_count
           FROM nb_lift;

QUANTILE_NUMBER PROBABILITY_THRESHOLD GAIN_CUMULATIVE QUANTILE_TOTAL_COUNT
--------------- --------------------- --------------- --------------------
              1            .989335775       .15034965                   55
              2            .980534911       .26048951                   55
              3            .968506098      .374125874                   55
              4            .958975196      .493006993                   55
              5            .946705997      .587412587                   55
              6            .927454174       .66958042                   55
              7            .904403627      .748251748                   55
              8            .836482525      .839160839                   55
             10            .500184953               1                   54
```

COMPUTE_LIFT_PART Procedure

The `COMPUTE_LIFT_PART` procedure computes lift and stores the results in a table in the user's schema. This procedure provides support to the computation of evaluation metrics per-partition for partitioned models.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. `COMPUTE_LIFT_PART` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT_PART` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT_PART` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

See Also:

Oracle Machine Learning for SQL Concepts for more details about Lift and test metrics for classification

"COMPUTE_LIFT Procedure"

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_CONFUSION_MATRIX_PART Procedure"

"COMPUTE_ROC Procedure"

"COMPUTE_ROC_PART Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT_PART (
      apply_result_table_name      IN VARCHAR2,
      target_table_name            IN VARCHAR2,
      case_id_column_name          IN VARCHAR2,
      target_column_name           IN VARCHAR2,
      lift_table_name              IN VARCHAR2,
      positive_target_value        IN VARCHAR2,
      score_column_name            IN VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name  IN VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name  IN VARCHAR2 DEFAULT 'PARTITION_NAME',
      num_quantiles                IN NUMBER   DEFAULT 10,
      cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
      apply_result_schema_name     IN VARCHAR2 DEFAULT NULL,
      target_schema_name           IN VARCHAR2 DEFAULT NULL,
      cost_matrix_schema_name      IN VARCHAR2 DEFAULT NULL,
      score_criterion_type         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-63 COMPUTE_LIFT_PART Procedure Parameters

Parameter	Description
`apply_result_table_name`	Table containing the predictions
`target_table_name`	Table containing the known target values from the test data
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`lift_table_name`	Table containing the Lift statistics. The table will be created by the procedure in the user's schema. The columns in the Lift table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate Lift. If the target column is a `NUMBER`, then you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, then the class with the lowest cost is predicted. The `score_criterion_type` parameter indicates whether probabilities or costs will be used for scoring. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure"). See the Usage Notes for additional information.
`score_partition_column_name`	Optional parameter indicating the column containing the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`num_quantiles`	Number of quantiles to be used in calculating Lift. The default is 10.
`cost_matrix_table_name`	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the `score_criterion_type` parameter is set to `COST`, then the costs will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
`apply_result_schema_name`	Schema of the apply results table If null, then the user's schema is assumed.

Table 62-63 (Cont.) COMPUTE_LIFT_PART Procedure Parameters

Parameter	Description
`target_schema_name`	Schema of the table containing the known targets If null, then the user's schema is assumed.
`cost_matrix_schema_name`	Schema of the cost matrix table, if one is provided If null, then the user's schema is assumed.
`score_criterion_type`	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the `score_criterion_column_name` parameter. The default value of `score_criterion_type` is `PROBABILITY`. To use costs as the scoring criterion, specify `COST`. If `score_criterion_type` is set to `COST` but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring. See the Usage Notes and the Examples.

Usage Notes

- The predictive information you pass to `COMPUTE_LIFT_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the Lift.
- Instead of passing a cost matrix to `COMPUTE_LIFT_PART`, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL `PREDICTION_COST` function to populate the score criterion column.
- The predictions that you pass to `COMPUTE_LIFT_PART` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- A cost matrix must have the columns described in Table 62-61.

Table 62-64 Columns in a Cost Matrix

Column Name	Data Type
`actual_target_value`	Type of the target column in the test data
`predicted_target_value`	Type of the predicted target in the test data. The type of the predicted target must be the same as the type of the actual target unless the predicted target has an associated reverse transformation.
`cost`	`NUMBER`

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more information about cost matrixes

- The table created by `COMPUTE_LIFT_PART` has the columns described in Table 62-62

Table 62-65 Columns in a COMPUTE_LIFT_PART Table

Column Name	Data Type
quantile_number	NUMBER
probability_threshold	NUMBER
gain_cumulative	NUMBER
quantile_total_count	NUMBER
quantile_target_count	NUMBER
percent_records_cumulative	NUMBER
lift_cumulative	NUMBER
target_density_cumulative	NUMBER
targets_cumulative	NUMBER
non_targets_cumulative	NUMBER
lift_quantile	NUMBER
target_density	NUMBER

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for details about the information in the Lift table

- When a cost matrix is passed to `COMPUTE_LIFT_PART`, the cost threshold is returned in the `probability_threshold` column of the Lift table.

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The example illustrates Lift based on probabilities. For examples that show computation based on costs, see "COMPUTE_CONFUSION_MATRIX Procedure".

For a partitioned model example, see "COMPUTE_CONFUSION_MATRIX_PART Procedure".

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute Lift as follows.

```
BEGIN
      DBMS_DATA_MINING.COMPUTE_LIFT_PART (
             apply_result_table_name     => 'nb_apply_results',
             target_table_name           => 'mining_data_test_v',
             case_id_column_name         => 'cust_id',
             target_column_name          => 'affinity_card',
             lift_table_name             => 'nb_lift',
             positive_target_value       =>  to_char(1),
             score_column_name           => 'PREDICTION',
             score_criterion_column_name => 'PROBABILITY',
             score_partition_column_name => 'PARTITITON_NAME',
             num_quantiles               =>  10,
             cost_matrix_table_name      =>  null,
             apply_result_schema_name    =>  null,
             target_schema_name          =>  null,
             cost_matrix_schema_name     =>  null,
             score_criterion_type        =>  'PROBABILITY');
END;
/
```

This query displays some of the statistics from the resulting Lift table.

```
SELECT quantile_number, probability_threshold, gain_cumulative,
          quantile_total_count
          FROM nb_lift;

QUANTILE_NUMBER PROBABILITY_THRESHOLD GAIN_CUMULATIVE QUANTILE_TOTAL_COUNT
--------------- --------------------- --------------- --------------------
              1            .989335775       .15034965                   55
              2            .980534911       .26048951                   55
              3            .968506098      .374125874                   55
              4            .958975196      .493006993                   55
              5            .946705997      .587412587                   55
              6            .927454174       .66958042                   55
              7            .904403627      .748251748                   55
              8            .836482525      .839160839                   55
             10            .500184953               1                   54
```

COMPUTE_ROC Procedure

This procedure computes the receiver operating characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. `COMPUTE_ROC` compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is 0.5. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an X-Y axis. The false positive rate is placed on the X axis. The true positive rate is placed on the Y axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

`COMPUTE_ROC` accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for more details about ROC and test metrics for classification
>
> "COMPUTE_CONFUSION_MATRIX Procedure"
>
> "COMPUTE_LIFT Procedure"

Syntax

```
DBMS_DATA_MINING.COMPUTE_ROC (
      roc_area_under_curve          OUT NUMBER,
      apply_result_table_name       IN  VARCHAR2,
      target_table_name             IN  VARCHAR2,
      case_id_column_name           IN  VARCHAR2,
      target_column_name            IN  VARCHAR2,
      roc_table_name                IN  VARCHAR2,
      positive_target_value         IN  VARCHAR2,
      score_column_name             IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name   IN  VARCHAR2 DEFAULT 'PROBABILITY',
      apply_result_schema_name      IN  VARCHAR2 DEFAULT NULL,
      target_schema_name            IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-66 COMPUTE_ROC Procedure Parameters

Parameter	Description
`roc_area_under_the_curve`	Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another.
`apply_result_table_name`	Table containing the predictions.

Table 62-66 (Cont.) COMPUTE_ROC Procedure Parameters

Parameter	Description
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`roc_table_name`	Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a `NUMBER`, you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is '`PREDICTION`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is '`PROBABILITY`', which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`apply_result_schema_name`	Schema of the apply results table. If null, the user's schema is assumed.
`target_schema_name`	Schema of the table containing the known targets. If null, the user's schema is assumed.

Usage Notes

- The predictive information you pass to `COMPUTE_ROC` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.
- The predictions that you pass to `COMPUTE_ROC` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- The table created by `COMPUTE_ROC` has the columns shown in Table 62-67.

Table 62-67 COMPUTE_ROC Output

Column	Datatype
probability	BINARY_DOUBLE
true_positives	NUMBER
false_negatives	NUMBER
false_positives	NUMBER
true_negatives	NUMBER
true_positive_fraction	NUMBER
false_positive_fraction	NUMBER

See Also:

Oracle Machine Learning for SQL Concepts for details about the output of `COMPUTE_ROC`

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

  Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```
SELECT case_id_column_name
       FROM apply_result_table_name
       WHERE probability > probability_threshold
       ORDER BY probability DESC;
```

- There are two approaches to identifying the most desirable probability threshold. Which approach you use depends on whether or not you know the relative cost of positive versus negative class prediction errors.

  If the costs are known, you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query like this.

```
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false_positives cost
    FROM ROC_table
  GROUP BY probability_threshold),
    minCost AS (
      SELECT min(cost) minCost
        FROM cost)
      SELECT max(probability_threshold)probability_threshold
        FROM cost, minCost
    WHERE cost = minCost;
```

  If relative costs are not well known, you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```
SELECT * FROM ROC_table
        ORDER BY probability_threshold;
```

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```
DECLARE
     v_area_under_curve NUMBER;
BEGIN
     DBMS_DATA_MINING.COMPUTE_ROC (
         roc_area_under_curve         => v_area_under_curve,
         apply_result_table_name      => 'nb_apply_results',
         target_table_name            => 'mining_data_test_v',
         case_id_column_name          => 'cust_id',
         target_column_name           => 'mining_data_test_v',
         roc_table_name               => 'nb_roc',
         positive_target_value        => '1',
         score_column_name            => 'PREDICTION',
         score_criterion_column_name => 'PROBABILITY');
     DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
     ROUND(v_area_under_curve,4));
END;
/
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212

 SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION
           FROM NB_ROC;

PROBABILITY  TRUE_POSITIVE_FRACTION  FALSE_POSITIVE_FRACTION
-----------  ----------------------  -----------------------
     .00000                       1                        1
     .50018              .826589595               .227902946
     .53851              .823699422               .221837088
     .54991              .820809249               .217504333
     .55628              .815028902               .215771231
     .55628              .817919075               .215771231
     .57563              .800578035               .214904679
     .57563              .812138728               .214904679
      .                   .                        .
      .                   .                        .
      .                   .                        .
```

COMPUTE_ROC_PART Procedure

The `COMPUTE_ROC_PART` procedure computes Receiver Operating Characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy. This procedure provides support to computation of evaluation metrics per-partition for partitioned models.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. `COMPUTE_ROC_PART` compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is `0.5`. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an x-y axis. The false positive rate is placed on the x-axis. The true positive rate is placed on the y-axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

`COMPUTE_ROC_PART` accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

See Also:

Oracle Machine Learning for SQL Concepts for more details about ROC and test metrics for Classification

"COMPUTE_ROC Procedure"

"COMPUTE_CONFUSION_MATRIX Procedure"

"COMPUTE_LIFT_PART Procedure"

"COMPUTE_LIFT Procedure"

Syntax

```
DBMS_DATA_MINING.compute_roc_part(
      roc_area_under_curve        OUT DM_NESTED_NUMERICALS,
      apply_result_table_name     IN  VARCHAR2,
      target_table_name           IN  VARCHAR2,
      case_id_column_name         IN  VARCHAR2,
      target_column_name          IN  VARCHAR2,
```

```
      roc_table_name              IN  VARCHAR2,
      positive_target_value       IN  VARCHAR2,
      score_column_name           IN  VARCHAR2 DEFAULT 'PREDICTION',
      score_criterion_column_name IN  VARCHAR2 DEFAULT 'PROBABILITY',
      score_partition_column_name IN  VARCHAR2 DEFAULT 'PARTITION_NAME',
      apply_result_schema_name    IN  VARCHAR2 DEFAULT NULL,
      target_schema_name          IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-68 COMPUTE_ROC_PART Procedure Parameters

Parameter	Description
`roc_area_under_the_curve`	Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive. The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another. The output argument is changed from `NUMBER` to `DM_NESTED_NUMERICALS`.
`apply_result_table_name`	Table containing the predictions.
`target_table_name`	Table containing the known target values from the test data.
`case_id_column_name`	Case ID column in the apply results table. Must match the case identifier in the targets table.
`target_column_name`	Target column in the targets table. Contains the known target values from the test data.
`roc_table_name`	Table containing the ROC output. The table will be created by the procedure in the user's schema. The columns in the ROC table are described in the Usage Notes.
`positive_target_value`	The positive class. This should be the class of interest, for which you want to calculate ROC. If the target column is a `NUMBER`, then you can use the `TO_CHAR()` operator to provide the value as a string.
`score_column_name`	Column containing the predictions in the apply results table. The default column name is `PREDICTION`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_criterion_column_name`	Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions. The default column name is `PROBABILITY`, which is the default name created by the `APPLY` procedure (See "APPLY Procedure").
`score_partition_column_name`	Optional parameter indicating the column which contains the name of the partition. This column slices the input test results such that each partition has independent evaluation matrices computed.
`apply_result_schema_name`	Schema of the apply results table. If null, then the user's schema is assumed.

Table 62-68 (Cont.) COMPUTE_ROC_PART Procedure Parameters

Parameter	Description
`target_schema_name`	Schema of the table containing the known targets. If null, then the user's schema is assumed.

Usage Notes

- The predictive information you pass to `COMPUTE_ROC_PART` may be generated using SQL `PREDICTION` functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.
- The predictions that you pass to `COMPUTE_ROC_PART` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
            case_id_column_name            VARCHAR2,
            score_column_name              VARCHAR2,
            score_criterion_column_name    VARCHAR2);
```

- The `COMPUTE_ROC_PART` table has the following columns:

Table 62-69 COMPUTE_ROC_PART Output

Column	Data Type
`probability`	`BINARY_DOUBLE`
`true_positives`	`NUMBER`
`false_negatives`	`NUMBER`
`false_positives`	`NUMBER`
`true_negatives`	`NUMBER`
`true_positive_fraction`	`NUMBER`
`false_positive_fraction`	`NUMBER`

> **See Also:**
>
> *Oracle Machine Learning for SQL Concepts* for details about the output of `COMPUTE_ROC_PART`

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

  Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```
SELECT case_id_column_name
       FROM apply_result_table_name
```

```
WHERE probability > probability_threshold
ORDER BY probability DESC;
```

- There are two approaches to identify the most desirable probability threshold. The approach you use depends on whether you know the relative cost of positive versus negative class prediction errors.

  If the costs are known, then you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query as follows:

```
WITH cost AS (
  SELECT probability_threshold, 20 * false_negatives + false_positives cost
    FROM ROC_table
  GROUP BY probability_threshold),
    minCost AS (
      SELECT min(cost) minCost
        FROM cost)
      SELECT max(probability_threshold)probability_threshold
        FROM cost, minCost
    WHERE cost = minCost;
```

  If relative costs are not well known, then you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```
SELECT * FROM ROC_table
         ORDER BY probability_threshold;
```

Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
    SELECT cust_id, t.prediction, t.probability
    FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```
DECLARE
     v_area_under_curve NUMBER;
BEGIN
     DBMS_DATA_MINING.COMPUTE_ROC_PART (
         roc_area_under_curve        => v_area_under_curve,
         apply_result_table_name     => 'nb_apply_results',
         target_table_name           => 'mining_data_test_v',
         case_id_column_name         => 'cust_id',
         target_column_name          => 'affinity_card',
         roc_table_name              => 'nb_roc',
         positive_target_value       => '1',
         score_column_name           => 'PREDICTION',
         score_criterion_column_name => 'PROBABILITY');
         score_partition_column_name => 'PARTITION_NAME'
     DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
     ROUND(v_area_under_curve,4));
```

```
END;
/
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212

 SELECT PROBABILITY, TRUE_POSITIVE_FRACTION, FALSE_POSITIVE_FRACTION
            FROM NB_ROC;

PROBABILITY  TRUE_POSITIVE_FRACTION  FALSE_POSITIVE_FRACTION
-----------  ----------------------  -----------------------
     .00000                       1                        1
     .50018              .826589595               .227902946
     .53851              .823699422               .221837088
     .54991              .820809249               .217504333
     .55628              .815028902               .215771231
     .55628              .817919075               .215771231
     .57563              .800578035               .214904679
     .57563              .812138728               .214904679
      .                    .                         .
      .                    .                         .
      .                    .                         .
```

CREATE_MODEL Procedure

This procedure creates an Oracle Machine Learning for SQL model with a given machine learning function.

Syntax

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

Parameters

Table 62-70   CREATE_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used. See the Usage Notes for model naming restrictions.
`mining_function`	The machine learning function. Values are listed in Table 62-3.
`data_table_name`	Table or view containing the build data
`case_id_column_name`	Case identifier column in the build data.
`target_column_name`	For supervised models, the target column in the build data. `NULL` for unsupervised models.

Table 62-70 (Cont.) CREATE_MODEL Procedure Parameters

Parameter	Description
settings_table_name	Table containing build settings for the model. NULL if there is no settings table (only default settings are used).
data_schema_name	Schema hosting the build data. If NULL, then the user's schema is assumed.
settings_schema_name	Schema hosting the settings table. If NULLthen the user's schema is assumed.

xform_list	A list of transformations to be used in addition to or instead of automatic transformations, depending on the value of the PREP_AUTO setting. (See "Automatic Data Preparation".)
	The datatype of xform_list is TRANSFORM_LIST, which consists of records of type TRANSFORM_REC. Each TRANSFORM_REC specifies the transformation information for a single attribute.
	`TYPE`
	`  TRANFORM_REC      IS RECORD (`
	`     attribute_name         VARCHAR2(4000),`
	`     attribute_subname      VARCHAR2(4000),`
	`     expression             EXPRESSION_REC,`
	`     reverse_expression     EXPRESSION_REC,`
	`     attribute_spec         VARCHAR2(4000));`
	The expression field stores a SQL expression for transforming the attribute. The reverse_expression field stores a SQL expression for reversing the transformation in model details and, if the attribute is a target, in the results of scoring. The SQL expressions are manipulated by routines in the DBMS_DATA_MINING_TRANSFORM package:
	• SET_EXPRESSION Procedure
	• GET_EXPRESSION Function
	• SET_TRANSFORM Procedure
	The attribute_spec field identifies individualized treatment for the attribute. See the Usage Notes for details.
	See Table 63-1for details about the TRANSFORM_REC type.

Usage Notes

1. You can use the attribute_spec field of the xform_list argument to identify an attribute as unstructured text or to disable Automatic Data Preparation for the attribute. The attribute_spec can have the following values:

   - TEXT: Indicates that the attribute contains unstructured text. The TEXT value may optionally be followed by POLICY_NAME, TOKEN_TYPE, MAX_FEATURES, and MIN_DOCUMENTS parameters.

     TOKEN_TYPE has the following possible values: NORMAL, STEM, THEME, SYNONYM, BIGRAM, STEM_BIGRAM. SYNONYM may be optionally followed by a thesaurus name in square brackets.

     MAX_FEATURES specifies the maximum number of tokens extracted from the text.

     MIN_DOCUMENTS specifies the minimal number of documents in which every selected token shall occur. (For information about creating a text policy, see CTX_DDL.CREATE_POLICY in *Oracle Text Reference*).

Oracle Machine Learning for SQL can process columns of VARCHAR2/CHAR, CLOB, BLOB, and BFILE as text. If the column is VARCHAR2 or CHAR and you do not specify TEXT, then OML4SQL processes the column as categorical data. If the column is CLOB, then OML4SQL processes it as text by default (You do not need to specify it as TEXT. However, you do need to provide an Oracle Text Policy in the settings). If the column is BLOB or BFILE, then you must specify it as TEXT, otherwise CREATE_MODEL returns an error.

If you specify TEXT for a nested column or for an attribute in a nested column, then CREATE_MODEL returns an error.

- NOPREP: Disables ADP for the attribute. When ADP is OFF, the NOPREP value is ignored.

  You can specify NOPREP for a nested column, but not for an attribute in a nested column. If you specify NOPREP for an attribute in a nested column when ADP is on, then CREATE_MODEL will return an error.

2. You can obtain information about a model by querying the Data Dictionary views.

```
ALL/USER/DBA_MINING_MODELS
ALL/USER/DBA_MINING_MODEL_ATTRIBUTES
ALL/USER/DBA_MINING_MODEL_SETTINGS
ALL/USER/DBA_MINING_MODEL_VIEWS
ALL/USER/DBA_MINING_MODEL_PARTITIONS
ALL/USER/DBA_MINING_MODEL_XFORMS
```

   You can obtain information about model attributes by querying the model details through model views. Refer to *Oracle Machine Learning for SQL User’s Guide*.

3. The naming rules for models are more restrictive than the naming rules for most database schema objects. A model name must satisfy the following additional requirements:

   - It must be 123 or fewer characters long.
   - It must be a nonquoted identifier. Oracle requires that nonquoted identifiers contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#); the initial character must be alphabetic. Oracle strongly discourages the use of the dollar sign and pound sign in nonquoted literals.

   Naming requirements for schema objects are fully documented in *Oracle Database SQL Language Reference*.

4. To build a partitioned model, you must provide additional settings.

   The setting for partitioning columns are as follows:

```
INSERT INTO settings_table VALUES (‘ODMS_PARTITION_COLUMNS’, ‘GENDER, 
AGE’);
```

   To set user-defined partition number for a model, the setting is as follows:

```
INSERT INTO settings_table VALUES ('ODMS_MAX_PARTITIONS’, '10’);
```

   The default value for maximum number of partitions is 1000.

5. By passing an xform_list to CREATE_MODEL, you can specify a list of transformations to be performed on the input data. If the PREP_AUTO setting is ON, the transformations are used in addition to the automatic transformations. If the PREP_AUTO setting is OFF, the specified transformations are the only ones implemented by the model. In both cases, transformation definitions are embedded in the model and run automatically whenever the

model is applied. See "Automatic Data Preparation". Other transforms that can be specified with `xform_list` include `FORCE_IN`. Refer to *Oracle Machine Learning for SQL User's Guide*.

Examples

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

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

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

You can display the model settings with the following query:

```
SELECT * FROM user_mining_model_settings
       WHERE model_name IN 'SVM_MODEL';

MODEL_NAME     SETTING_NAME            SETTING_VALUE                 SETTING
-------------  ----------------------  ----------------------------  -------
SVM_MODEL      ALGO_NAME               ALGO_SUPPORT_VECTOR_MACHINES  INPUT

SVM_MODEL      SVMS_STD_DEV            3.004524                      DEFAULT
SVM_MODEL      PREP_AUTO               ON                            INPUT
SVM_MODEL      SVMS_COMPLEXITY_FACTOR  1.887389                      DEFAULT
SVM_MODEL      SVMS_KERNEL_FUNCTION    SVMS_LINEAR                      DEFAULT
SVM_MODEL      SVMS_CONV_TOLERANCE     .001                          DEFAULT
```

The following is an example of querying a model view instead of the older `GEL_MODEL_DETAILS_SVM` routine.

```
SELECT target_value, attribute_name, attribute_value, coefficient   FROM
DM$VLSVM_MODEL;
```

The second example creates an anomaly detection model. Anomaly detection uses SVM classification without a target. This example uses the same settings table created for the SVM classification model in the first example.

```
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'anomaly_detect_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name     => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name  => null,
    settings_table_name => 'svm_model_settings');
END;
/
```

This query shows that the models created in these examples are the only ones in your schema.

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

MODEL_NAME              MINING_FUNCTION      ALGORITHM
----------------------  -------------------- ------------------------------
SVM_MODEL               CLASSIFICATION       SUPPORT_VECTOR_MACHINES
ANOMALY_DETECT_MODEL    CLASSIFICATION       SUPPORT_VECTOR_MACHINES
```

This query shows that only the SVM classification model has a target.

```
SELECT model_name, attribute_name, attribute_type, target
       FROM user_mining_model_attributes
       WHERE target = 'YES';

MODEL_NAME          ATTRIBUTE_NAME   ATTRIBUTE_TYPE     TARGET
------------------  ---------------  -----------------  ------
SVM_MODEL           AFFINITY_CARD    CATEGORICAL         YES
```

CREATE_MODEL2 Procedure

The `CREATE_MODEL2` procedure is an alternate procedure to the `CREATE_MODEL` procedure, which enables creating a model without extra persistence stages. In the `CREATE_MODEL` procedure, the input is a table or a view and if such an object is not already present, the user must create it. By using the `CREATE_MODEL2` procedure, the user does not need to create such transient database objects.

Syntax

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

Parameters

Table 62-71 CREATE_MODEL2 Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form [`schema_name`.]`model_name`. If you do not specify a schema, then the current schema is used. See the Usage Notes, CREATE_MODEL Procedure for model naming restrictions.
`mining_function`	The machine learning function. Values are listed in DBMS_DATA_MINING — Machine Learning Function Settings.
`data_query`	A query which provides training data for building the model.
`set_list`	Specifies the `SETTING_LIST` `SETTING_LIST` is a table of CLOB index by `VARCHAR2(30)`; Where the index is the setting name and the CLOB is the setting value for that name.
`case_id_column_name`	Case identifier column in the build data.
`target_column_name`	For supervised models, the target column in the build data. `NULL` for unsupervised models.
`xform_list`	Refer to CREATE_MODEL Procedure.

Usage Notes

Refer to CREATE_MODEL Procedure for Usage Notes.

Examples

The following example uses the Support Vector Machine algorithm.

```
declare
 v_setlst DBMS_DATA_MINING.SETTING_LIST;

BEGIN
  v_setlst(dbms_data_mining.algo_name) :=
dbms_data_mining.algo_support_vector_machines;
  v_setlst(dbms_data_mining.prep_auto) := dbms_data_mining.prep_auto_on;

DBMS_DATA_MINING.CREATE_MODEL2(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_query          => 'select * from mining_data_build_v',
    data_table_name     => 'mining_data_build_v',
    case_id_column_name=> 'cust_id',
    target_column_name => 'affinity_card',
    set_list            => v_setlst,
    case_id_column_name=> 'cust_id',
    target_column_name => 'affinity_card');
END;
/
```

Create Model Using Registration Information

Create model function fetches the setting information from JSON object.

Usage Notes

If an algorithm is registered, user can create model using the registered algorithm name. Since all R scripts and default setting values are already registered, providing the value through the setting table is not necessary. This makes the use of this algorithm easier.

Examples

The first example builds a Classification model using the GLM algorithm.

```
CREATE TABLE GLM_RDEMO_SETTINGS_CL (

   setting_name  VARCHAR2(30),
   setting_value VARCHAR2(4000));
   BEGIN
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         ('ALGO_EXTENSIBLE_LANG', 'R');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         (dbms_data_mining.ralg_registration_algo_name, 't1');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
        (dbms_data_mining.odms_formula,
        'AGE + EDUCATION + HOUSEHOLD_SIZE + OCCUPATION');
        INSERT INTO GLM_RDEMO_SETTINGS_CL VALUES
         ('RALG_PARAMETER_FAMILY',   'binomial(logit)' );
   END;
   /
     BEGIN
          DBMS_DATA_MINING.CREATE_MODEL(
          model_name                    =>    'GLM_RDEMO_CLASSIFICATION',
          mining_function               =>     dbms_data_mining.classification,
          data_table_name               =>    'mining_data_build_v',
          case_id_column_name           =>    'CUST_ID',
          target_column_name            =>    'AFFINITY_CARD',
          settings_table_name           =>    'GLM_RDEMO_SETTINGS_CL');
      END;
      /
```

DROP_ALGORITHM Procedure

This function is used to drop the registered algorithm information.

Syntax

```
DBMS_DATA_MINING.DROP_ALGORITHM (algorithm_name  IN  VARCHAR2(30),
                                 cascade         IN  BOOLEAN default FALSE)
```

Parameters

Table 62-72   DROP_ALGORITHM Procedure Parameters

Parameter	Description
algorithm_name	Name of the algorithm.

Table 62-72 (Cont.) DROP_ALGORITHM Procedure Parameters

Parameter	Description
`cascade`	If the cascade option is `TRUE`, all the models with this algorithms are forced to drop. There after, the algorithm is dropped. The default value is `FALSE`.

Usage Note

- To drop a machine learning model, you must be the owner or you must have the `RQADMIN` privilege. See *Oracle Machine Learning for SQL User's Guide* for information about privileges for machine learning.
- Make sure a model is not built on the algorithm, then drop the algorithm from the system table.
- If you try to drop an algorithm with a model built on it, then an error is displayed.

DROP_PARTITION Procedure

Syntax

```
DBMS_DATA_MINING.DROP_PARTITION (
      model_name                  IN VARCHAR2,
      partition_name              IN VARCHAR2);
```

Parameters

Table 62-73 DROP_PARTITION Procedure Parameters

Parameters	Description
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Name of the partition that must be dropped.

DROP_MODEL Procedure

This procedure deletes the specified machine learning model.

Syntax

```
DBMS_DATA_MINING.DROP_MODEL (model_name IN VARCHAR2,
                             force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 62-74 DROP_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Table 62-74 (Cont.) DROP_MODEL Procedure Parameters

Parameter	Description
`force`	Forces the machine learning model to be dropped even if it is invalid. A machine learning model may be invalid if a serious system error interrupted the model build process.

Usage Note

To drop a machine learning model, you must be the owner or you must have the `DROP ANY MINING MODEL` privilege. See *Oracle Data Mining User's Guide* for information about privileges for Oracle Machine Learning for SQL.

Example

You can use the following command to delete a valid machine learning model named `nb_sh_clas_sample` that exists in your schema.

```
BEGIN
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'nb_sh_clas_sample');
END;
/
```

EXPORT_MODEL Procedure

This procedure exports the specified machine learning models to a dump file set.

To import the models from the dump file set, use the IMPORT_MODEL Procedure. `EXPORT_MODEL` and `IMPORT_MODEL` use Oracle Data Pump technology.

When Oracle Data Pump is used to export/import an entire schema or database, the machine learning models in the schema or database are included. However, `EXPORT_MODEL` and `IMPORT_MODEL` are the only utilities that support the export/import of individual models.

> **See Also:**
>
> *Oracle Database Utilities* for information about Oracle Data Pump
>
> *Oracle Machine Learning for SQL User's Guide* for more information about exporting and importing machine learning models

Syntax

```
DBMS_DATA_MINING.EXPORT_MODEL (
      filename          IN VARCHAR2,
      directory         IN VARCHAR2,
      model_filter      IN VARCHAR2 DEFAULT NULL,
      filesize          IN VARCHAR2 DEFAULT NULL,
      operation         IN VARCHAR2 DEFAULT NULL,
      remote_link       IN VARCHAR2 DEFAULT NULL,
      jobname           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-75 EXPORT_MODEL Procedure Parameters

Parameter	Description
`filename`	Name of the dump file set to which the models should be exported. The name must be unique within the schema. The dump file set can contain one or more files. The number of files in a dump file set is determined by the size of the models being exported (both metadata and data) and a specified or estimated maximum file size. You can specify the file size in the `filesize` parameter, or you can use the `operation` parameter to cause Oracle Data Pump to estimate the file size. If the size of the models to export is greater than the maximum file size, one or more additional files are created. When the export operation completes successfully, the name of the dump file set is automatically expanded to *`filename01.dmp`*, even if there is only one file in the dump set. If there are additional files, they are named sequentially as *`filename02.dmp`*, *`filename03.dmp`*, and so forth.
`directory`	Name of a pre-defined directory object that specifies where the dump file set should be created. The exporting user must have read/write privileges on the directory object and on the file system directory that it identifies. See *Oracle Database SQL Language Reference* for information about directory objects.
`model_filter`	Optional parameter that specifies which model or models to export. If you do not specify a value for `model_filter`, all models in the schema are exported. You can also specify `NULL` (the default) or `'ALL'` to export all models. You can export individual models by name and groups of models based on machine learning function or algorithm. For instance, you could export all regression models or all Naive Bayes models. Examples are provided in Table 62-76.
`filesize`	Optional parameter that specifies the maximum size of a file in the dump file set. The size may be specified in bytes, kilobytes (K), megabytes (M), or gigabytes (G). The default size is 50 MB. If the size of the models to export is larger than `filesize`, one or more additional files are created within the dump set. See the description of the `filename` parameter for more information.
`operation`	Optional parameter that specifies whether or not to estimate the size of the files in the dump set. By default the size is not estimated and the value of the `filesize` parameter determines the size of the files. You can specify either of the following values for `operation`: • `'EXPORT'` — Export all or the specified models. (Default) • `'ESTIMATE'` — Estimate the size of the exporting models.
`remote_link`	Optional parameter that specifies the name of a database link to a remote system. The default value is `NULL`. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for `remote_link`, you can export the models in the remote database. The `EXP_FULL_DATABASE` role is required for exporting the remote models. The `EXP_FULL_DATABASE` privilege, the `CREATE DATABASE LINK` privilege, and other privileges may also be required.

Table 62-75 (Cont.) EXPORT_MODEL Procedure Parameters

Parameter	Description
`jobname`	Optional parameter that specifies the name of the export job. By default, the name has the form *`username`*`_exp_`*`nnnn`*, where *`nnnn`* is a number. For example, a job name in the `SCOTT` schema might be `SCOTT_exp_134`. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the export job, named *`jobname.log`*, is created in the same directory as the dump file set.

Usage Notes

The `model_filter` parameter specifies which models to export. You can list the models by name, or you can specify all models that have the same machine learning function or algorithm. You can query the `USER_MINING_MODELS` view to list the models in your schema.

```
SQL> describe user_mining_models
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 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)
```

Examples of model filters are provided in Table 62-76.

Table 62-76 Sample Values for the Model Filter Parameter

Sample Value	Meaning
`'mymodel'`	Export the model named `mymodel`
`'name= ''mymodel'''`	Export the model named `mymodel`
`'name IN (''mymodel2'',''mymodel3'')'`	Export the models named `mymodel2` and `mymodel3`
`'ALGORITHM_NAME = ''NAIVE_BAYES'''`	Export all Naive Bayes models. See Table 62-5 for a list of algorithm names.
`'FUNCTION_NAME =''CLASSIFICATION'''`	Export all classification models. See Table 62-3 for a list of machine learning functions.

Examples

1. The following statement exports all the models in the `oml_user3` schema to a dump file set called `models_out` in the directory `$ORACLE_HOME/rdbms/log`. This directory is mapped to a directory object called `DATA_PUMP_DIR`. The `oml_user3` user has read/write access to the directory and to the directory object.

   ```
   SQL>execute dbms_data_mining.export_model ('models_out', 'DATA_PUMP_DIR');
   ```

   You can exit SQL*Plus and list the resulting dump file and log file.

```
SQL>EXIT
>cd $ORACLE_HOME/rdbms/log
>ls
>oml_user3_exp_1027.log  models_out01.dmp
```

2. The following example uses the same directory object and is run by the same user. This example exports the models called `NMF_SH_SAMPLE` and `SVMR_SH_REGR_SAMPLE` to a different dump file set in the same directory.

```
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL ( 'models2_out', 'DATA_PUMP_DIR',
            'name in (''NMF_SH_SAMPLE'', ''SVMR_SH_REGR_SAMPLE'')');
SQL>EXIT
>cd $ORACLE_HOME/rdbms/log
>ls
>oml_user3_exp_1027.log  models_out01.dmp
 oml_user3_exp_924.log  models2_out01.dmp
```

3. The following examples show how to export models with specific algorithm and machine learning function names.

```
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('algo.dmp','DM_DUMP',
        'ALGORITHM_NAME IN (''O_CLUSTER'',''GENERALIZED_LINEAR_MODEL'',
        ''SUPPORT_VECTOR_MACHINES'',''NAIVE_BAYES'')');

SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL('func.dmp', 'DM_DUMP',
        'FUNCTION_NAME IN (CLASSIFICATION,CLUSTERING,FEATURE_EXTRACTION)');
```

EXPORT_SERMODEL Procedure

This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

When exporting a model in serialized format, the user must pass in an empty `BLOB` locator and specify the model name to be exported. If the model is partitioned, the user can optionally select an individual partition to export, otherwise all partitions are exported. The returned `BLOB` contains the content that can be deployed.

Syntax

```
DBMS_DATA_MINING.EXPORT_SERMODEL (
      model_data     IN OUT NOCOPY BLOB,
      model_name     IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-77   EXPORT_SERMODEL Procedure Parameters

Parameter	Description
`model_data`	Provides serialized model data.
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Name of the partition that must be exported.

Examples

The following statement exports all of the models in a serialized format.

```
DECLARE
 v_blob blob;
BEGIN
 dbms_lob.createtemporary(v_blob, FALSE);
 dbms_data_mining.export_sermodel(v_blob, 'MY_MODEL');
-- save v_blob somewhere (e.g., bfile, etc.)
 dbms_lob.freetemporary(v_blob);
END;
/
```

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for more information about exporting and importing machine learning models

FETCH_JSON_SCHEMA Procedure

User can fetch and read JSON schema from the `ALL_MINING_ALGORITHMS` view. This function returns the pre-registered JSON schema for R extensible algorithms.

Syntax

```
DBMS_DATA_MINING.FETCH_JSON_SCHEMA RETURN CLOB;
```

Parameters

Table 62-78 FETCH_JSON_SCHEMA Procedure Parameters

Parameter	Description
`RETURN`	This function returns the pre-registered JSON schema for R extensibility. The default value is `CLOB`.

Usage Note

If a user wants to register a new algorithm using the algorithm registration function, they must fetch and follow the pre-registered JSON schema using this function, when they create the required JSON object metadata, and then pass it to the registration function.

GET_ASSOCIATION_RULES Function

The `GET_ASSOCIATION_RULES` function returns the rules produced by an association model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

You can specify filtering criteria to GET_ASSOCIATION_RULES to return a subset of the rules. Filtering criteria can improve the performance of the table function. If the number of rules is large, then the greatest performance improvement will result from specifying the topn parameter.

Syntax

```
DBMS_DATA_MINING.get_association_rules(
      model_name        IN VARCHAR2,
      topn              IN NUMBER DEFAULT NULL,
      rule_id           IN INTEGER DEFAULT NULL,
      min_confidence    IN NUMBER DEFAULT NULL,
      min_support       IN NUMBER DEFAULT NULL,
      max_rule_length   IN INTEGER DEFAULT NULL,
      min_rule_length   IN INTEGER DEFAULT NULL,
      sort_order        IN ORA_MINING_VARCHAR2_NT DEFAULT NULL,
      antecedent_items  IN DM_ITEMS DEFAULT NULL,
      consequent_items  IN DM_ITEMS DEFAULT NULL,
      min_lift          IN NUMBER DEFAULT NULL,
      partition_name    IN VARCHAR2 DEFAULT NULL)
  RETURN DM_Rules PIPELINED;
```

Parameters

Table 62-79 GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
model_name	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used. This is the only required parameter of GET_ASSOCIATION_RULES. All other parameters specify optional filters on the rules to return.
topn	Returns the *n* top rules ordered by confidence and then support, both descending. If you specify a sort order, then the top *n* rules are derived after the sort is performed. If topn is specified and no maximum or minimum rule length is specified, then the only columns allowed in the sort order are RULE_CONFIDENCE and RULE_SUPPORT. If topn is specified and a maximum or minimum rule length *is* specified, then RULE_CONFIDENCE, RULE_SUPPORT, and NUMBER_OF_ITEMS are allowed in the sort order.
rule_id	Identifier of the rule to return. If you specify a value for rule_id, do not specify values for the other filtering parameters.
min_confidence	Returns the rules with confidence greater than or equal to this number.
min_support	Returns the rules with support greater than or equal to this number.
max_rule_length	Returns the rules with a length less than or equal to this number. Rule length refers to the number of items in the rule (See NUMBER_OF_ITEMS in Table 62-80). For example, in the rule A=>B (if A, then B), the number of items is 2. If max_rule_length is specified, then the NUMBER_OF_ITEMS column is permitted in the sort order.
min_rule_length	Returns the rules with a length greater than or equal to this number. See max_rule_length for a description of rule length. If min_rule_length is specified, then the NUMBER_OF_ITEMS column is permitted in the sort order.

Table 62-79 (Cont.) GET_ASSOCIATION_RULES Function Parameters

Parameter	Description
sort_order	Sorts the rules by the values in one or more of the returned columns. Specify one or more column names, each followed by ASC for ascending order or DESC for descending order. (See Table 62-80 for the column names.) For example, to sort the result set in descending order first by the NUMBER_OF_ITEMS column, then by the RULE_CONFIDENCE column, you must specify: `ORA_MINING_VARCHAR2_NT('NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC')` If you specify topn, the results will vary depending on the sort order. By default, the results are sorted by Confidence in descending order, then by Support in descending order.
antecedent_items	Returns the rules with these items in the antecedent.
consequent_items	Returns the rules with this item in the consequent.
min_lift	Returns the rules with lift greater than or equal to this number.
partition_name	Specifies a partition in a partitioned model.

Return Values

The object type returned by GET_ASSOCIATION_RULES is described in Table 62-80. For descriptions of each field, see the Usage Notes.

Table 62-80 GET_ASSOCIATION RULES Function Return Values

Return Value	Description

DM_RULES	A set of rows of type DM_RULE. The rows have the following columns:
	`(rule_id INTEGER,`
	`antecedent DM_PREDICATES,`
	`consequent DM_PREDICATES,`
	`rule_support NUMBER,`
	`rule_confidence NUMBER,`
	`rule_lift NUMBER,`
	`antecedent_support NUMBER,`
	`consequent_support NUMBER,`
	`number_of_items INTEGER )`

DM_PREDICATES	The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`conditional_operator CHAR(2)/*=,<>,<,>,<=,>=*/,`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000),`
	`attribute_support NUMBER,`
	`attribute_confidence NUMBER)`

Usage Notes

1. This table function pipes out rows of type `DM_RULES`. For information on machine learning data types and piped output from table functions, see "Datatypes".
2. The columns returned by `GET_ASSOCIATION_RULES` are described as follows:

Column in DM_RULES	Description
`rule_id`	Unique identifier of the rule

`antecedent`	The independent condition in the rule. When this condition exists, the dependent condition in the consequent also exists.
	The condition is a combination of attribute values called a predicate (`DM_PREDICATE`). The predicate specifies a condition for each attribute. The condition may specify equality (=), inequality (<>), greater than (>), less than (<), greater than or equal to (>=), or less than or equal to (<=) a given value.
	`Support` and `Confidence` for each attribute condition in the antecedent is returned in the predicate. Support is the number of transactions that satisfy the antecedent. Confidence is the likelihood that a transaction will satisfy the antecedent.
	Note: The occurrence of the attribute as a `DM_PREDICATE` indicates the presence of the item in the transaction. The actual value for `attribute_num_value` or `attribute_str_value` is meaningless. For example, the following predicate indicates that 'Mouse Pad' is present in the transaction *even though* the attribute value is `NULL`.
	`DM_PREDICATE('PROD_NAME',`
	`                'Mouse Pad', '= ', NULL, NULL, NULL, NULL))`

`consequent`	The dependent condition in the rule. This condition exists when the antecedent exists. The consequent, like the antecedent, is a predicate (`DM_PREDICATE`). Support and confidence for each attribute condition in the consequent is returned in the predicate. Support is the number of transactions that satisfy the consequent. Confidence is the likelihood that a transaction will satisfy the consequent.
`rule_support`	The number of transactions that satisfy the rule.
`rule_confidence`	The likelihood of a transaction satisfying the rule.
`rule_lift`	The degree of improvement in the prediction over random chance when the rule is satisfied.
`antecedent_support`	The ratio of the number of transactions that satisfy the antecedent to the total number of transactions.
`consequent_support`	The ratio of the number of transactions that satisfy the consequent to the total number of transactions.
`number_of_items`	The total number of attributes referenced in the antecedent and consequent of the rule.

Examples

The following example demonstrates an association model build followed by several invocations of the `GET_ASSOCIATION_RULES` table function:

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
SELECT *
```

```
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

-- build an AR model
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'market_settings');
END;
/
-- View the (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model'));
```

In the previous example, you view all rules. To view just the top 20 rules, use the following statement.

```
-- View the top 20 (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model', 20));
```

The following query uses the association model `AR_SH_SAMPLE`.

```
SELECT * FROM TABLE (
   DBMS_DATA_MINING.GET_ASSOCIATION_RULES (
      'AR_SH_SAMPLE', 10, NULL, 0.5, 0.01, 2, 1,
         ORA_MINING_VARCHAR2_NT (
         'NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC', 'RULE_SUPPORT DESC'),
         DM_ITEMS(DM_ITEM('CUSTPRODS', 'Mouse Pad', 1, NULL),
                  DM_ITEM('CUSTPRODS', 'Standard Mouse', 1, NULL)),
         DM_ITEMS(DM_ITEM('CUSTPRODS', 'Extension Cable', 1, NULL))));
```

The query returns three rules, shown as follows:

```
13  DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL),
       DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
    .15532      .84393   2.7075     .18404     .3117   2

11  DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
    .18085      .56291   1.8059     .32128     .3117   1

9   DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL))
    DM_PREDICATES(
       DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
      .17766    .55116   1.7682     .32234     .3117   1
```

See Also:

Table 62-80 for the `DM_RULE` column data types.

GET_FREQUENT_ITEMSETS Function

The `GET_FREQUENT_ITEMSETS` function returns a set of rows that represent the frequent itemsets from an association model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead..

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

For a detailed description of frequent itemsets, consult *Oracle Machine Learning for SQL Concepts*.

Syntax

```
DBMS_DATA_MINING.get_frequent_itemsets(
      model_name IN VARCHAR2,
      topn IN NUMBER DEFAULT NULL,
      max_itemset_length IN NUMBER DEFAULT NULL,
      partition_name     IN VARCHAR2 DEFAULT NULL)
  RETURN DM_ItemSets PIPELINED;
```

Parameters

Table 62-81 GET_FREQUENT_ITEMSETS Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`topn`	When not `NULL`, return the top *n* rows ordered by support in descending order
`max_itemset_length`	Maximum length of an item set.
`partition_name`	Specifies a partition in a partitioned model.

Note:

The `partition_name` columns applies only when the model is partitioned.

Return Values

Table 62-82 GET_FREQUENT_ITEMSETS Function Return Values

Return Value	Description

DM_ITEMSETS	A set of rows of type DM_ITEMSET. The rows have the following columns:
	`(partition_name VARCHAR2(128)`
	`itemsets_id NUMBER,`
	`items DM_ITEMS,`
	`support NUMBER,`
	`number_of_items NUMBER)`
	Note: The partition_name columns applies only when the model is partitioned.
	The items column returns a nested table of type DM_ITEMS. The rows have type DM_ITEM:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_num_value NUMBER,`
	`attribute_str_value VARCHAR2(4000))`

Usage Notes

This table function pipes out rows of type DM_ITEMSETS. For information on machine learning data types and piped output from table functions, see "Data Types".

Examples

The following example demonstrates an association model build followed by an invocation of GET_FREQUENT_ITEMSETS table function from Oracle SQL.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS

    SELECT *

  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

/* build a AR model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name            => 'market_model',
  function              => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name       => 'market_build',
  case_id_column_name   => 'item_id',
  target_column_name    => NULL,
```

```
  settings_table_name  => 'market_settings');
END;
/

-- View the (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model'));
```

In the example above, you view all itemsets. To view just the top 20 itemsets, use the following statement:

```
-- View the top 20 (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
  FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model', 20));
```

GET_MODEL_COST_MATRIX Function

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

The GET_MODEL_COST_MATRIX function is replaced by the `DM$VC` prefixed view, Scoring Cost Matrix. The cost matrix used when building a Decision Tree is made available by the `DM$VM` prefixed view, Decision Tree build cost matrix.

Refer to Model Detail View for Classification Algorithm.

The `GET_MODEL_COST_MATRIX` function returns the rows of a cost matrix associated with the specified model.

By default, this function returns the scoring cost matrix that was added to the model with the `ADD_COST_MATRIX` procedure. If you wish to obtain the cost matrix used to create a model, specify `cost_matrix_type_create` as the `matrix_type`. See Table 62-83.

See also ADD_COST_MATRIX Procedure.

Syntax

```
DBMS_DATA_MINING.GET_MODEL_COST_MATRIX (
      model_name                IN VARCHAR2,
      matrix_type               IN VARCHAR2 DEFAULT cost_matrix_type_score)
      partition_name            IN VARCHAR2 DEFAULT NULL);
RETURN DM_COST_MATRIX PIPELINED;
```

Parameters

Table 62-83 GET_MODEL_COST_MATRIX Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`matrix_type`	The type of cost matrix. `COST_MATRIX_TYPE_SCORE` — cost matrix used for scoring. (Default.) `COST_MATRIX_TYPE_CREATE` — cost matrix used to create the model (Decision Tree only).
`partition_name`	Name of the partition in a partitioned model

Return Values

Table 62-84 GET_MODEL_COST_MATRIX Function Return Values

Return Value	Description

`DM_COST_MATRIX`	A set of rows of type `DM_COST_ELEMENT`. The rows have the following columns:
	`actual VARCHAR2(4000), NUMBER, predicted`
	`VARCHAR2(4000), cost NUMBER)`

Usage Notes

Only Decision Tree models can be built with a cost matrix. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See Table 62-7.

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, you can use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one.

The `GET_MODEL_COST_MATRIX` may return either the build or scoring cost matrix defined for a model or model partition.

If you do not specify a partitioned model name, then an error is displayed.

Example

This example returns the scoring cost matrix associated with the Naive Bayes model `NB_SH_CLAS_SAMPLE`.

```
column actual format a10
column predicted format a10
SELECT *
    FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
    ORDER BY predicted, actual;

ACTUAL     PREDICTED   COST
---------- ---------- -----
0          0            .00
1          0            .75
0          1            .25
1          1            .00
```

GET_MODEL_DETAILS_AI Function

The `GET_MODEL_DETAILS_AI` function returns a set of rows that provide the details of an attribute importance model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

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

```
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN dm_ranked_attributes pipelined;
```

Parameters

Table 62-85 GET_MODEL_DETAILS_AI Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 62-86 GET_MODEL_DETAILS_AI Function Return Values

Return Value	Description

`DM_RANKED_ATTRIBUTES`	A set of rows of type `DM_RANKED_ATTRIBUTE`. The rows have the following columns:
	`(attribute_name VARCHAR2(4000,`
	`attribute_subname VARCHAR2(4000),`
	`importance_value NUMBER,`
	`rank NUMBER(38))`

Examples

The following example returns model details for the attribute importance model `AI_SH_sample`, which was created by the sample program `dmaidemo.sql`.

```
SELECT attribute_name, importance_value, rank
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_AI('AI_SH_sample'))
    ORDER BY RANK;

ATTRIBUTE_NAME                          IMPORTANCE_VALUE       RANK
---------------------------------------- ---------------- ----------
HOUSEHOLD_SIZE                                .151685183          1
CUST_MARITAL_STATUS                           .145294546          2
YRS_RESIDENCE                                  .07838928          3
AGE                                           .075027496          4
Y_BOX_GAMES                                   .063039952          5
EDUCATION                                     .059605314          6
HOME_THEATER_PACKAGE                          .056458722          7
OCCUPATION                                    .054652937          8
CUST_GENDER                                   .035264741          9
BOOKKEEPING_APPLICATION                       .019204751         10
PRINTER_SUPPLIES                                       0         11
OS_DOC_SET_KANJI                              -.00050013         12
FLAT_PANEL_MONITOR                            -.00509564         13
BULK_PACK_DISKETTES                           -.00540822         14
COUNTRY_NAME                                  -.01201116         15
CUST_INCOME_LEVEL                             -.03951311         16
```

GET_MODEL_DETAILS_EM Function

The `GET_MODEL_DETAILS_EM` function returns a set of rows that provide statistics about the clusters produced by an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

By default, the EM algorithm groups components into high-level clusters, and `GET_MODEL_DETAILS_EM` returns only the high-level clusters with their hierarchies. Alternatively, you can configure EM model to disable the grouping of components into high-level clusters. In this case, `GET_MODEL_DETAILS_EM` returns the components themselves as clusters with their hierarchies. See Table 62-12.

Syntax

```
DBMS_DATA_MINING.get_model_details_em(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      attribute_subname  VARCHAR2 DEFAULT NULL,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 62-87 GET_MODEL_DETAILS_EM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise, the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned

Table 62-87 (Cont.) GET_MODEL_DETAILS_EM Function Parameters

Parameter	Description
`attribute_subname`	The name of a nested attribute. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the column and *`attribute_subname`* is the name of the nested attribute in that column. If the attribute is not nested, then `attribute_subname` is null.
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *`n`* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *`topn`*, then, up to *`n`* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. For information on Oracle Machine Learning for SQL data types and return values for Clustering algorithms piped output from table functions, see "Data Types".
2. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.
3. When cluster statistics are disabled (`EMCS_CLUSTER_STATISTICS` is set to `EMCS_CLUS_STATS_DISABLE`), `GET_MODEL_DETAILS_EM` does not return centroids, histograms, or rules. Only taxonomy (hierarchy) and cluster counts are returned.
4. When the `partition_name` is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

GET_MODEL_DETAILS_EM_COMP Function

he `GET_MODEL_DETAILS_EM_COMP` table function returns a set of rows that provide details about the parameters of an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_em_comp(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_EM_COMPONENT_SET PIPELINED;
```

Parameters

Table 62-88 GET_MODEL_DETAILS_EM_COMP Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model to retrieve details for.

Return Values

Table 62-89 GET_MODEL_DETAILS_EM_COMP Function Return Values

Return Value	Description

`DM_EM_COMPONENT_SET`	A set of rows of type `DM_EM_COMPONENT`. The rows have the following columns:
	`(info_type VARCHAR2(30),`
	`component_id NUMBER,`
	`cluster_id NUMBER,`
	`attribute_name VARCHAR2(4000),`
	`covariate_name VARCHAR2(4000),`
	`attribute_value VARCHAR2(4000),`
	`value NUMBER )`

Usage Notes

1. This table function pipes out rows of type `DM_EM_COMPONENT`. For information on Oracle Machine Learning for SQL data types and piped output from table functions, see "Data Types".

   The columns in each row returned by `GET_MODEL_DETAILS_EM_COMP` are described as follows:

Column in DM_EM_COMPONENT	Description
`info_type`	The type of information in the row. The following information types are supported: • `cluster` • `prior` • `mean` • `covariance` • `frequency`
`component_id`	Unique identifier of a component
`cluster_id`	Unique identifier of the high-level leaf cluster for each component
`attribute_name`	Name of an original attribute or a derived feature ID. The derived feature ID is used in models built on data with nested columns. The derived feature definitions can be obtained from the GET_MODEL_DETAILS_EM_PROJ Function.

Column in DM_EM_COMPONENT	Description
`covariate_name`	Name of an original attribute or a derived feature ID used in variance/covariance definition
`attribute_value`	Categorical value or bin interval for binned numerical attributes
`value`	Encodes different information depending on the value of `info_type`, as follows: • `cluster` — The value field is `NULL` • `prior` — The value field returns the component prior • `mean` — The value field returns the mean of the attribute specified in `attribute_name` • `covariance` — The value field returns the covariance of the attributes specified in `attribute_name` and `covariate_name`. Using the same attribute in `attribute_name` and `covariate_name`, returns the variance. • `frequency`— The value field returns the multivalued Bernoulli frequency parameter for the attribute/value combination specified by `attribute_name` and `attribute_value` See Usage Note 2 for details.

2. The following table shows which fields are used for each `info_type`. The blank cells represent `NULL`s.

info_type	component_id	cluster_id	attribute_name	covariate_name	attribute_value	value
cluster	X	X				
prior	X	X				X
mean	X	X	X			X
covariance	X	X	X	X		X
frequency	X	X	X		X	X

3. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.
4. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

GET_MODEL_DETAILS_EM_PROJ Function

The `GET_MODEL_DETAILS_EM_PROJ` function returns a set of rows that provide statistics about the projections produced by an expectation maximization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

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

```
     partition_name IN VARCHAR2 DEFAULT NULL)
 RETURN DM_EM_PROJECTION_SET PIPELINED;
```

Parameters

Table 62-90 GET_MODEL_DETAILS_EM_PROJ Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 62-91 GET_MODEL_DETAILS_EM_PROJ Function Return Values

Return Value	Description

`DM_EM_PROJECTION_SET`	A set of rows of type `DM_EM_PROJECTION`. The rows have the following columns:
	`(feature_name VARCHAR2(4000),`
	` attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` attribute_value VARCHAR2(4000),`
	` coefficient NUMBER )`
	See Usage Notes for details.

Usage Notes

1. This table function pipes out rows of type `DM_EM_PROJECTION`. For information on machine learning data types and piped output from table functions, see "Datatypes".

   The columns in each row returned by `GET_MODEL_DETAILS_EM_PROJ` are described as follows:

Column in DM_EM_PROJECTION	Description
`feature_name`	Name of a derived feature. The feature maps to the attribute_name returned by the GET_MODEL_DETAILS_EM Function.
`attribute_name`	Name of a column in the build data
`attribute_subname`	Subname in a nested column
`attribute_value`	Categorical value
`coefficient`	Projection coefficient. The representation is sparse; only the non-zero coefficients are returned.

2. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

   The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information.

3. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_GLM Function

The GET_MODEL_DETAILS_GLM function returns the coefficient statistics for a generalized linear model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

The same set of statistics is returned for both linear and logistic regression, but statistics that do not apply to the machine learning function are returned as NULL. For more details, see the Usage Notes.

Syntax

```
DBMS_DATA_MINING.get_model_details_glm(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_GLM_Coeff_Set PIPELINED;
```

Parameters

Table 62-92 GET_MODEL_DETAILS_GLM Function Parameters

Parameter	Description
model_name	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
partition_name	Specifies a partition in a partitioned model

Return Values

Table 62-93 GET_MODEL_DETAILS_GLM Return Values

Return Value	Description

DM_GLM_COEFF_SET	A set of rows of type DM_GLM_COEFF. The rows have the following columns:
	`(class VARCHAR2(4000),`
	`attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_value VARCHAR2(4000),`
	`feature_expression VARCHAR2(4000),`
	`coefficient NUMBER,`
	`std_error NUMBER,`
	`test_statistic NUMBER,`
	`p_value NUMBER,`
	`VIF NUMBER,`
	`std_coefficient NUMBER,`
	`lower_coeff_limit NUMBER,`
	`upper_coeff_limit NUMBER,`
	`exp_coefficient BINARY_DOUBLE,`
	`exp_lower_coeff_limit BINARY_DOUBLE,`
	`exp_upper_coeff_limit BINARY_DOUBLE)`

`GET_MODEL_DETAILS_GLM` returns a row of statistics for each attribute and one extra row for the intercept, which is identified by a null value in the attribute name. Each row has the `DM_GLM_COEFF` data type. The statistics are described in Table 62-94.

Table 62-94 DM_GLM_COEFF Data Type Description

Column	Description
class	The non-reference target class for logistic regression. The model is built to predict the probability of this class. The other class (the reference class) is specified in the model setting `GLMS_REFERENCE_CLASS_NAME`. See Table 62-19. For Linear Regression, `class` is null.
attribute_name	The attribute name when there is no subname, or first part of the attribute name when there is a subname. The value of `attribute_name` is also the name of the column in the case table that is the source for this attribute. For the intercept, `attribute_name` is null. Intercepts are equivalent to the bias term in SVM models.
attribute_subname	The name of an attribute in a nested table. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the nested column in the case table that is the source for this attribute. If the attribute is not nested, then `attribute_subname` is null. If the attribute is an intercept, then both the `attribute_name` and the `attribute_subname` are null.

Table 62-94 (Cont.) DM_GLM_COEFF Data Type Description

Column	Description
`attribute_value`	The value of the attribute (categorical attribute only). For numeric attributes, `attribute_value` is null.
`feature_expression`	The feature name constructed by the algorithm when feature generation is enabled and higher-order features are found. If feature selection is not enabled, then the feature name is simply the fully-qualified attribute name (*`attribute_name.attribute_subname`* if the attribute is in a nested column). For categorical attributes, the algorithm constructs a feature name that has the following form: *`fully-qualified_attribute_name.attribute_value`* For numeric attributes, the algorithm constructs a name for the higher-order feature by taking the product of the resulting values: (*`attrib1`*)*(*`attrib2`*))*...... where *`attrib1`* and *`attrib2`* are fully-qualified attribute names.
`coefficient`	The linear coefficient estimate.
`std_error`	Standard error of the coefficient estimate.
`test_statistic`	For linear regression, the t-value of the coefficient estimate. For logistic regression, the Wald chi-square value of the coefficient estimate.
`p-value`	Probability of the `test_statistic`. Used to analyze the significance of specific attributes in the model.
`VIF`	Variance Inflation Factor. The value is zero for the intercept. For logistic regression, `VIF` is null. VIF is not computed if the solver is Cholesky.
`std_coefficient`	Standardized estimate of the coefficient.
`lower_coeff_limit`	Lower confidence bound of the coefficient.
`upper_coeff_limit`	Upper confidence bound of the coefficient.
`exp_coefficient`	Exponentiated coefficient for logistic regression. For linear regression, `exp_coefficient` is null.
`exp_lower_coeff_limit`	Exponentiated coefficient for lower confidence bound of the coefficient for logistic regression. For linear regression, `exp_lower_coeff_limit` is null.
`exp_upper_coeff_limit`	Exponentiated coefficient for upper confidence bound of the coefficient for logistic regression. For linear regression, `exp_lower_coeff_limit` is null.

Usage Notes

Not all statistics are necessarily returned for each coefficient. Statistics will be null if:

- They do not apply to the machine learning function. For example, `exp_coefficient` does not apply to linear regression.
- They cannot be computed from a theoretical standpoint. For information on ridge regression, see Table 62-19.
- They cannot be computed because of limitations in system resources.
- Their values would be infinity.

- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns some of the model details for the GLM regression model `GLMR_SH_Regr_sample`.

```
SET line 120
SET pages 99
column attribute_name format a30
column attribute_subname format a20
column attribute_value format a20
col coefficient format 990.9999
col std_error format 990.9999
SQL> SELECT * FROM
(SELECT attribute_name, attribute_value, coefficient, std_error
  FROM DM$VDGLMR_SH_REGR_SAMPLE order by 1,2)
WHERE rownum < 11;

ATTRIBUTE_NAME                 ATTRIBUTE_VALUE      COEFFICIENT    STD_ERROR
------------------------------ -------------------- ----------- ---------
AFFINITY_CARD                                          -0.5797       0.5283
BOOKKEEPING_APPLICATION                              -0.4689       3.8872
BULK_PACK_DISKETTES                                  -0.9819       2.5430
COUNTRY_NAME                     Argentina              -1.2020       1.1876
COUNTRY_NAME                     Australia              -0.0071       5.1146
COUNTRY_NAME                     Brazil                  5.2931       1.9233
COUNTRY_NAME                     Canada                   4.0191        2.4108
COUNTRY_NAME                     China                   0.8706       3.5889
COUNTRY_NAME                     Denmark              -2.9822        3.1803
COUNTRY_NAME                     France                  -1.1044        7.1811
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_GLOBAL Function

The `GET_MODEL_DETAILS_GLOBAL` function returns statistics about the model as a whole. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Global details are available for Generalized Linear Models, Association Rules, Singular Value Decomposition, and Expectation Maximization. There are new Global model views which show global information for all algorithms. Oracle recommends that users leverage the views instead. Refer to Model Details View Global.

Syntax

```
DBMS_DATA_MINING.get_model_details_global(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_model_global_details PIPELINED;
```

Parameters

Table 62-95 GET_MODEL_DETAILS_GLOBAL Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 62-96 GET_MODEL_DETAILS_GLOBAL Function Return Values

Return Value	Description

`DM_MODEL_GLOBAL_DETAILS`	A collection of rows of type `DM_MODEL_GLOBAL_DETAIL`. The rows have the following columns:
	`(global_detail_name VARCHAR2(30),`
	` global_detail_value NUMBER)`

Examples

The following example returns the global model details for the GLM regression model `GLMR_SH_Regr_sample`.

```
SELECT *
  FROM TABLE(dbms_data_mining.get_model_details_global(
              'GLMR_SH_Regr_sample'))
ORDER BY global_detail_name;
GLOBAL_DETAIL_NAME             GLOBAL_DETAIL_VALUE
------------------------------ -------------------
ADJUSTED_R_SQUARE                        .731412557
AIC                                        5931.814
COEFF_VAR                                18.1711243
CORRECTED_TOTAL_DF                             1499
CORRECTED_TOT_SS                         278740.504
DEPENDENT_MEAN                               38.892
ERROR_DF                                       1433
ERROR_MEAN_SQUARE                        49.9440956
ERROR_SUM_SQUARES                        71569.8891
F_VALUE                                  62.8492452
GMSEP                                     52.280819
HOCKING_SP                               .034877162
J_P                                      52.1749319
MODEL_CONVERGED                                   1
MODEL_DF                                         66
MODEL_F_P_VALUE                                   0
MODEL_MEAN_SQUARE                        3138.94871
MODEL_SUM_SQUARES                        207170.615
NUM_PARAMS                                       67
NUM_ROWS                                       1500
ROOT_MEAN_SQ                             7.06711367
R_SQ                                     .743238288
SBIC                                     6287.79977
VALID_COVARIANCE_MATRIX                           1
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_KM Function

The `GET_MODEL_DETAILS_KM` function returns a set of rows that provide the details of a *k*-means clustering model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

You can provide input to `GET_MODEL_DETAILS_KM` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, then `GET_MODEL_DETAILS_KM` returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.get_model_details_km(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      attribute_subname  VARCHAR2 DEFAULT NULL,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 62-97 GET_MODEL_DETAILS_KM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned

Table 62-97 (Cont.) GET_MODEL_DETAILS_KM Function Parameters

Parameter	Description
`attribute_subname`	The name of a nested attribute. The full name of a nested attribute has the form: *`attribute_name.attribute_subname`* where *`attribute_name`* is the name of the column and *`attribute_subname`* is the name of the nested attribute in that column. If the attribute is not nested, `attribute_subname` is null.
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *`n`* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *`topn`*, then up to *`n`* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. The table function pipes out rows of type `DM_CLUSTERS`. For information on machine learning data types and Return Value for Clustering Algorithms piped output from table functions, see "Data Types".
2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the *k*-means clustering model `KM_SH_Clus_sample`.

```
SELECT T.id           clu_id,
       T.record_count rec_cnt,
       T.parent       parent,
       T.tree_level   tree_level,
       T.dispersion   dispersion
  FROM (SELECT *
          FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_KM(
                     'KM_SH_Clus_sample'))
        ORDER BY id) T
 WHERE ROWNUM < 6;

    CLU_ID    REC_CNT     PARENT TREE_LEVEL DISPERSION
---------- ---------- ---------- ---------- ----------
         1       1500                     1  5.9152211
         2        638          1          2 3.98458982
         3        862          1          2 5.83732097
         4        376          3          3 5.05192137
         5        486          3          3 5.42901522
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_NB Function

The `GET_MODEL_DETAILS_NB` function returns a set of rows that provide the details of a naive Bayes model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_nb(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_NB_Details PIPELINED;
```

Parameters

Table 62-98 GET_MODEL_DETAILS_NB Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 62-99 GET_MODEL_DETAILS_NB Function Return Values

Return Value	Description

`DM_NB_DETAILS`	A set of rows of type `DM_NB_DETAIL`. The rows have the following columns:
	`(target_attribute_name VARCHAR2(30),`
	` target_attribute_str_value VARCHAR2(4000),`
	` target_attribute_num_value NUMBER,`
	` prior_probability NUMBER,`
	` conditionals DM_CONDITIONALS)`
	The `conditionals` column of `DM_NB_DETAIL` returns a nested table of type `DM_CONDITIONALS`. The rows, of type `DM_CONDITIONAL`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` attribute_str_value VARCHAR2(4000),`
	` attribute_num_value NUMBER,`
	` conditional_probability NUMBER)`

Usage Notes

- The table function pipes out rows of type `DM_NB_DETAILS`. For information on machine learning data types and piped output from table functions, see "Data Types".
- When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following query is from the sample program `dmnbdemo.sql`. It returns model details about the model `NB_SH_Clas_sample`. For information about the sample programs, see *Oracle Machine Learning for SQL User's Guide*.

The query creates labels from the bin boundary tables that were used to bin the training data. It replaces the attribute values with the labels. For numeric bins, the labels are (*lower_boundary*,*upper_boundary*]; for categorical bins, the label matches the value it represents. (This method of categorical label representation will only work for cases where one value corresponds to one bin.) The target was not binned.

```
WITH
    bin_label_view AS (
    SELECT col, bin, (DECODE(bin,'1','[','(') || lv || ',' || val || ']') label
      FROM (SELECT col,
                   bin,
                   LAST_VALUE(val) OVER (
                   PARTITION BY col ORDER BY val
                   ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) lv,
                   val
             FROM nb_sh_sample_num)
   UNION ALL
   SELECT col, bin, val label
     FROM nb_sh_sample_cat
   ),
   model_details AS (
   SELECT T.target_attribute_name                                        tname,
          NVL(TO_CHAR(T.target_attribute_num_value,T.target_attribute_str_value)) tval,
          C.attribute_name                                               pname,
          NVL(L.label, NVL(C.attribute_str_value, C.attribute_num_value)) pval,
          T.prior_probability                                           priorp,
          C.conditional_probability                                      condp
     FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NB('NB_SH_Clas_sample')) T,
          TABLE(T.conditionals) C,
          bin_label_view L
    WHERE C.attribute_name = L.col (+) AND
          (NVL(C.attribute_str_value,C.attribute_num_value) = L.bin(+))
   ORDER BY 1,2,3,4,5,6
   )
   SELECT tname, tval, pname, pval, priorp, condp
     FROM model_details
    WHERE ROWNUM < 11;

TNAME          TVAL PNAME                     PVAL          PRIORP   CONDP
-------------- ---- ------------------------- ------------- ------- -------
AFFINITY_CARD  0    AGE                       (24,30]         .6500   .1714
AFFINITY_CARD  0    AGE                       (30,35]         .6500   .1509
AFFINITY_CARD  0    AGE                       (35,40]         .6500   .1125
AFFINITY_CARD  0    AGE                       (40,46]         .6500   .1134
AFFINITY_CARD  0    AGE                       (46,53]         .6500   .1071
AFFINITY_CARD  0    AGE                       (53,90]         .6500   .1312
AFFINITY_CARD  0    AGE                       [17,24]         .6500   .2134
AFFINITY_CARD  0    BOOKKEEPING_APPLICATION   0               .6500   .1500
AFFINITY_CARD  0    BOOKKEEPING_APPLICATION   1               .6500   .8500
AFFINITY_CARD  0    BULK_PACK_DISKETTES       0               .6500   .3670
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_NMF Function

The `GET_MODEL_DETAILS_NMF` function returns a set of rows that provide the details of a non-negative matrix factorization model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_nmf(
      model_name IN VARCHAR2,
      partition_name VARCHAR2 DEFAULT NULL)
   RETURN DM_NMF_Feature_Set PIPELINED;
```

Parameters

Table 62-100 GET_MODEL_DETAILS_NMF Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 62-101 GET_MODEL_DETAILS_NMF Function Return Values

Return Value	Description

`DM_NMF_FEATURE_SET`	A set of rows of `DM_NMF_FEATURE`. The rows have the following columns:
	`(feature_id NUMBER,`
	` mapped_feature_id VARCHAR2(4000),`
	` attribute_set DM_NMF_ATTRIBUTE_SET)`
	The `attribute_set` column of `DM_NMF_FEATURE` returns a nested table of type `DM_NMF_ATTRIBUTE_SET`. The rows, of type `DM_NMF_ATTRIBUTE`, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` attribute_value VARCHAR2(4000),`
	` coefficient NUMBER)`

Usage Notes

- The table function pipes out rows of type `DM_NMF_FEATURE_SET`. For information on machine learning data types and piped output from table functions, see "Data Types".
- When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the feature extraction model `NMF_SH_Sample`.

```
SELECT * FROM (
SELECT F.feature_id,
       A.attribute_name,
       A.attribute_value,
       A.coefficient
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF('NMF_SH_Sample')) F,
       TABLE(F.attribute_set) A
ORDER BY feature_id,attribute_name,attribute_value
) WHERE ROWNUM < 11;

FEATURE_ID ATTRIBUTE_NAME          ATTRIBUTE_VALUE          COEFFICIENT
--------- ----------------------- ---------------- -------------------
        1 AFFINITY_CARD                                .051208078859308
        1 AGE                                         .0390513260041573
        1 BOOKKEEPING_APPLICATION                     .0512734004239326
        1 BULK_PACK_DISKETTES                          .232471260895683
        1 COUNTRY_NAME            Argentina          .00766817464479959
        1 COUNTRY_NAME            Australia         .000157637881096675
        1 COUNTRY_NAME            Brazil              .0031409632415604
        1 COUNTRY_NAME            Canada             .00144213099311427
        1 COUNTRY_NAME            China             .000102279310968754
        1 COUNTRY_NAME            Denmark           .000242424084307513
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_OC Function

The `GET_MODEL_DETAILS_OC` function returns a set of rows that provide the details of an O-cluster clustering model. The rows are an enumeration of the clustering patterns generated during the creation of the model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

You can provide input to `GET_MODEL_DETAILS_OC` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, then `GET_MODEL_DETAILS_OC` returns all the information about the model.

Syntax

```
DBMS_DATA_MINING.get_model_details_oc(
      model_name VARCHAR2,
      cluster_id NUMBER   DEFAULT NULL,
      attribute  VARCHAR2 DEFAULT NULL,
      centroid   NUMBER   DEFAULT 1,
      histogram  NUMBER   DEFAULT 1,
      rules      NUMBER   DEFAULT 2,
      topn_attributes NUMBER DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN dm_clusters PIPELINED;
```

Parameters

Table 62-102 GET_MODEL_DETAILS_OC Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`cluster_id`	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
`attribute`	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise, the details for all attributes are returned
`centroid`	This parameter accepts the following values: • 1: Details about centroids are returned (default) • 0: Details about centroids are not returned
`histogram`	This parameter accepts the following values: • 1: Details about histograms are returned (default) • 0: Details about histograms are not returned
`rules`	This parameter accepts the following values: • 2: Details about rules are returned (default) • 1: Rule summaries are returned • 0: No information about rules is returned
`topn_attributes`	Restricts the number of attributes returned in the centroid, histogram, and rules objects. Only the *`n`* attributes with the highest confidence values in the rules are returned. If the number of attributes included in the rules is less than *`topn`*, then up to *`n`* additional attributes in alphabetical order are returned. If both the `attribute` and `topn_attributes` parameters are specified, then `topn_attributes` is ignored.
`partition_name`	Specifies a partition in a partitioned model.

Usage Notes

1. For information about machine learning data types and return values for clustering algorithms piped output from table functions, see "Data Types".
2. When the value is NULL for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the clustering model `OC_SH_Clus_sample`.

For each cluster in this example, the split predicate indicates the attribute and the condition used to assign records to the cluster's children during model build. It provides an important piece of information on how the population within a cluster can be divided up into two smaller clusters.

```
SELECT clu_id, attribute_name, op, s_value
     FROM (SELECT a.id clu_id, sp.attribute_name, sp.conditional_operator op,
                  sp.attribute_str_value s_value
              FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_OC(
                    'OC_SH_Clus_sample')) a,
```

```
                  TABLE(a.split_predicate) sp
           ORDER BY a.id, op, s_value)
     WHERE ROWNUM < 11;

     CLU_ID ATTRIBUTE_NAME       OP S_VALUE
----------- -------------------- ---------------------------------
          1 OCCUPATION           IN ?
          1 OCCUPATION           IN Armed-F
          1 OCCUPATION           IN Cleric.
          1 OCCUPATION           IN Crafts
          2 OCCUPATION           IN ?
          2 OCCUPATION           IN Armed-F
          2 OCCUPATION           IN Cleric.
          3 OCCUPATION           IN Exec.
          3 OCCUPATION           IN Farming
          3 OCCUPATION           IN Handler
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_SETTINGS Function

The `GET_MODEL_SETTINGS` function returns the settings used to build the given model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

Syntax

```
FUNCTION get_model_settings(model_name IN VARCHAR2)
  RETURN DM_Model_Settings PIPELINED;
```

Parameters

Table 62-103 GET_MODEL_SETTINGS Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Return Values

Table 62-104 GET_MODEL_SETTINGS Function Return Values

Return Value	Description

`DM_MODEL_SETTINGS`	A set of rows of type `DM_MODEL_SETTINGS`. The rows have the following columns:
	`DM_MODEL_SETTINGS TABLE OF SYS.DM_MODEL_SETTING`
	`Name                   Type`
	`---------------------- --------------------`
	`SETTING_NAME           VARCHAR2(30)`
	`SETTING_VALUE          VARCHAR2(4000)`

Usage Notes

1. This table function pipes out rows of type `DM_MODEL_SETTINGS`. For information on machine learning data types and piped output from table functions, see "DBMS_DATA_MINING Datatypes".
2. The setting names/values include both those specified by the user and any defaults assigned by the build process.

Examples

The following example returns model model settings for an example naive Bayes model.

```
SETTING_NAME                   SETTING_VALUE
------------------------------ ------------------------------
ALGO_NAME                       ALGO_NAIVE_BAYES
PREP_AUTO                       ON
ODMS_MAX_PARTITIONS             1000
NABS_SINGLETON_THRESHOLD       0
CLAS_WEIGHTS_BALANCED          OFF
NABS_PAIRWISE_THRESHOLD        0
ODMS_PARTITION_COLUMNS         GENDER,Y_BOX_GAMES
ODMS_MISSING_VALUE_TREATMENT   ODMS_MISSING_VALUE_AUTO
ODMS_SAMPLING                  ODMS_SAMPLING_DISABLE

9 rows selected.
```

Related Topics

- *Oracle Database Reference*

GET_MODEL_SIGNATURE Function

The `GET_MODEL_SIGNATURE` function returns the list of columns from the build input table that were used by the build process to train the model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

Syntax

```
FUNCTION get_model_signature (model_name IN VARCHAR2)
RETURN DM_Model_Signature PIPELINED;
```

Parameters

Table 62-105 GET_MODEL_SIGNATURE Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Return Values

Table 62-106 GET_MODEL_SIGNATURE Function Return Values

Return Value	Description

`DM_MODEL_SIGNATURE`	A set of rows of type `DM_MODEL_SIGNATURE`. The rows have the following columns:
	`DM_MODEL_SIGNATURE TABLE OF SYS.DM_MODEL_SIGNATURE_ATTRIBUTE`
	`Name                  Type`
	`------------------    --------------------`
	`ATTRIBUTE_NAME        VARCHAR2(130)`
	`ATTRIBUTE_TYPE        VARCHAR2(106)`

Usage Notes

1. This table function pipes out rows of type `DM_MODEL_SIGNATURE`. For information on machine learning data types and piped output from table functions, see "DBMS_DATA_MINING Datatypes".
2. The signature names or types include only those attributes used by the build process.

Examples

The following example returns model settings for an example naive Bayes model.

```
ATTRIBUTE_NAME                 ATTRIBUTE_TYPE
------------------------------ ------------------
AGE                            NUMBER
ANNUAL_INCOME                  NUMBER
AVERAGE___ITEMS_PURCHASED      NUMBER
BOOKKEEPING_APPLICATION        NUMBER
BULK_PACK_DISKETTES            NUMBER
BULK_PURCH_AVE_AMT             NUMBER
DISABLE_COOKIES                NUMBER
EDUCATION                      VARCHAR2
FLAT_PANEL_MONITOR             NUMBER
GENDER                         VARCHAR2
HOME_THEATER_PACKAGE           NUMBER
HOUSEHOLD_SIZE                 VARCHAR2
MAILING_LIST                   NUMBER
MARITAL_STATUS                 VARCHAR2
NO_DIFFERENT_KIND_ITEMS        NUMBER
OCCUPATION                     VARCHAR2
OS_DOC_SET_KANJI               NUMBER
PETS                           NUMBER
PRINTER_SUPPLIES               NUMBER
PROMO_RESPOND                  NUMBER
SHIPPING_ADDRESS_COUNTRY       VARCHAR2
SR_CITIZEN                     NUMBER
TOP_REASON_FOR_SHOPPING        VARCHAR2
WKS_SINCE_LAST_PURCH           NUMBER
WORKCLASS                      VARCHAR2
YRS_RESIDENCE                  NUMBER
Y_BOX_GAMES                    NUMBER

27 rows selected.
```

Related Topics

- *Oracle Database Reference*

GET_MODEL_DETAILS_SVD Function

The `GET_MODEL_DETAILS_SVD` function returns a set of rows that provide the details of a singular value decomposition model. Oracle recommends to use model details view settings. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

Refer to Model Details View for Singular Value Decomposition.

Syntax

```
DBMS_DATA_MINING.get_model_details_svd(
      model_name IN VARCHAR2,
      matrix_type IN VARCHAR2 DEFAULT NULL,
      partition_name VARCHAR2 DEFAULT NULL)
   RETURN DM_SVD_MATRIX_Set PIPELINED;
```

Parameters

Table 62-107 GET_MODEL_DETAILS_SVD Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`matrix_type`	Specifies which of the three SVD matrix types to return. Values are: `U`, `S`, `V`, and `NULL`. When `matrix_type` is null (default), all matrices are returned. The U matrix is only computed when the `SVDS_U_MATRIX_OUTPUT` setting is enabled. It is not computed by default. If the model does not contain U matrices and you set `matrix_type` to `U`, an empty set of rows is returned. See Table 62-27.
`partition_name`	A partition in a partitioned model.

Return Values

Table 62-108 GET_MODEL_DETAILS_SVD Function Return Values

Return Value	Description

`DM_SVD_MATRIX_SET`	A set of rows of type `DM_SVD_MATRIX`. The rows have the following columns:
	`(matrix_type CHAR(1),`
	` feature_id NUMBER,`
	` mapped_feature_id VARCHAR2(4000),`
	` attribute_name VARCHAR2(4000),`
	` attribute_subname VARCHAR2(4000),`
	` case_id VARCHAR2(4000),`
	` value NUMBER,`
	` variance NUMBER,`
	` pct_cum_variance NUMBER)`
	See Usage Notes for details.

Usage Notes

1. This table function pipes out rows of type `DM_SVD_MATRIX`. For information on machine learning data types and piped output from table functions, see "Data Types".

   The columns in each row returned by `GET_MODEL_DETAILS_SVD` are described as follows:

Column in DM_SVD_MATRIX_SET	Description
`matrix_type`	The type of matrix. Possible values are **S**, **V**, and **U**. This field is never null.
`feature_id`	The feature that the matrix entry refers to.
`mapped_feature_id`	A descriptive name for the feature.
`attribute_name`	Column name in the **V** matrix component bases. This field is null for the **S** and **U** matrices.
`attribute_subname`	Subname in the **V** matrix component bases. This is relevant only in the case of a nested column. This field is null for the **S** and **U** matrices.
`case_id`	Unique identifier of the row in the build data described by the **U** matrix projection. This field is null for the **S** and **V** matrices.
`value`	The matrix entry value.
`variance`	The variance explained by a component. It is non-null only for **S** matrix entries. This column is non-null only for **S** matrix entries and for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` and the build data is centered, either manually or because the setting `dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.
`pct_cum_variance`	The percent cumulative variance explained by the components thus far. The components are ranked by the explained variance in descending order. This column is non-null only for **S** matrix entries and for SVD models with setting `dbms_data_mining.svds_scoring_mode` set to `dbms_data_mining.svds_scoring_pca` and the build data is centered, either manually or because the setting `dbms_data_mining.prep_auto` is set to `dbms_data_mining.prep_auto_on`.

2. The output of `GET_MODEL_DETAILS` is in sparse format. Zero values are not returned. Only the diagonal elements of the **S** matrix, the non-zero coefficients in the **V** matrix bases, and the non-zero **U** matrix projections are returned.

   There is one exception: If the data row does not produce non-zero **U** Matrix projections, the case ID for that row is returned with `NULL` for the `feature_id` and `value`. This is done to avoid losing any records from the original data.

3. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

4. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the preferred partition name.

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_SVM Function

The `GET_MODEL_DETAILS_SVM` function returns a set of rows that provide the details of a linear support vector machines (SVM) model. If invoked for nonlinear SVM, it returns `ORA-40215`. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views in *Oracle Machine Learning for SQL User's Guide*.

In linear SVM models, only nonzero coefficients are stored. This reduces storage and speeds up model loading. As a result, if an attribute is missing in the coefficient list returned by `GET_MODEL_DETAILS_SVM`, then the coefficient of this attribute should be interpreted as zero.

Syntax

```
DBMS_DATA_MINING.get_model_details_svm(
      model_name   VARCHAR2,
      reverse_coef NUMBER DEFAULT 0,
      partition_name VARCHAR2 DEFAULT NULL)
  RETURN DM_SVM_Linear_Coeff_Set PIPELINED;
```

Parameters

Table 62-109 GET_MODEL_DETAILS_SVM Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`reverse_coef`	Whether or not `GET_MODEL_DETAILS_SVM` should transform the attribute coefficients using the original attribute transformations. When `reverse_coef` is set to 0 (default), `GET_MODEL_DETAILS_SVM` returns the coefficients directly from the model without applying transformations. When `reverse_coef` is set to 1, `GET_MODEL_DETAILS_SVM` transforms the coefficients and bias by applying the normalization shifts and scales that were generated using automatic data preparation. See Usage Note 4.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 62-110 GET_MODEL_DETAILS_SVM Function Return Values

Return Value	Description

DM_SVM_LINEAR_COEFF_SET	A set of rows of type DM_SVM_LINEAR_COEFF. The rows have the following columns:
	`(class VARCHAR2(4000),`
	`attribute_set DM_SVM_ATTRIBUTE_SET)`
	The attribute_set column returns a nested table of type DM_SVM_ATTRIBUTE_SET. The rows, of type DM_SVM_ATTRIBUTE, have the following columns:
	`(attribute_name VARCHAR2(4000),`
	`attribute_subname VARCHAR2(4000),`
	`attribute_value VARCHAR2(4000),`
	`coefficient NUMBER)`
	See Usage Notes.

Usage Notes

1. This table function pipes out rows of type `DM_SVM_LINEAR_COEFF`. For information on machine learning data types and piped output from table functions, see "Data Types".
2. The `class` column of `DM_SVM_LINEAR_COEFF` contains classification target values. For SVM Regression models, `class` is null. For each classification target value, a set of coefficients is returned. For binary classification, one-class classification, and regression models, only a single set of coefficients is returned.
3. The `attribute_value` column in `DM_SVM_ATTRIBUTE_SET` is used for categorical attributes.
4. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

   The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information. If you want `GET_MODEL_DETAILS_SVM` to transform the coefficients such that they relate to the original attributes, set the `reverse_coef` parameter to 1.
5. When the value is `NULL` for a partitioned model, an exception is thrown. When the value is not null, it must contain the desired partition name.

Examples

The following example returns model details for the SVM classification model `SVMC_SH_Clas_sample`, which was created by the sample program `dmsvcdem.sql`. For information about the sample programs, see *Oracle Machine Learning for SQL User's Guide*.

```
WITH
  mod_dtls AS (
  SELECT *
    FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('SVMC_SH_Clas_sample'))
  ),
  model_details AS (
```

```
  SELECT D.class, A.attribute_name, A.attribute_value, A.coefficient
    FROM mod_dtls D,
          TABLE(D.attribute_set) A
    ORDER BY D.class, ABS(A.coefficient) DESC
  )
  SELECT class, attribute_name aname, attribute_value aval, coefficient coeff
    FROM model_details
    WHERE ROWNUM < 11;

CLASS      ANAME                     AVAL                      COEFF
---------- ------------------------- ------------------------- -----
1                                                              -2.85
1          BOOKKEEPING_APPLICATION                              1.11
1          OCCUPATION                Other                      -.94
1          HOUSEHOLD_SIZE            4-5                         .88
1          CUST_MARITAL_STATUS       Married                     .82
1          YRS_RESIDENCE                                         .76
1          HOUSEHOLD_SIZE            6-8                        -.74
1          OCCUPATION                Exec.                       .71
1          EDUCATION                 11th                       -.71
1          EDUCATION                 Masters                     .63
```

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

GET_MODEL_DETAILS_XML Function

This function returns an XML object that provides the details of a decision tree model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. Use model detail views instead.

See Model Detail Views for Decision Tree in *Oracle Machine Learning for SQL User's Guide*.

Syntax

```
DBMS_DATA_MINING.get_model_details_xml(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN XMLType;
```

Parameters

Table 62-111 GET_MODEL_DETAILS_XML Function Parameters

Parameter	Description
`model_name`	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model.

Return Values

Table 62-112 GET_MODEL_DETAILS_XML Function Return Value

Return Value	Description
`XMLTYPE`	The XML definition for the decision tree model. See "XMLTYPE" for details. The XML definition conforms to the Data Mining Group Predictive Model Markup Language (PMML) version 2.1 specification. The specification is available at `https://dmg.org`. If a nested attribute is used as a splitter, the attribute will appear in the XML document as field="'<column_name>'.<subname>", as opposed to the non-nested attributes which appear in the document as field="<column_name>". **Note:** The column names are surrounded by single quotes and a period separates the column_name from the subname. The rest of the document style remains unchanged.

Usage Notes

Special characters that cannot be displayed by Oracle XML are converted to '#'.

Examples

The following statements in SQL*Plus return the details of the decision tree model `dt_sh_clas_sample`.

Note: The "&quot" characters you will see in the XML output are a result of SQL*Plus behavior. To display the XML in proper format, cut and past it into a file and open the file in a browser.

```
column dt_details format a320
SELECT
 dbms_data_mining.get_model_details_xml('dt_sh_clas_sample')
 AS DT_DETAILS
FROM dual;


DT_DETAILS
--------------------------------------------------------------------------------
<PMML version="2.1">
  <Header copyright="Copyright (c) 2004, Oracle Corporation. All rights
      reserved."/>
  <DataDictionary numberOfFields="9">
    <DataField name="AFFINITY_CARD" optype="categorical"/>
    <DataField name="AGE" optype="continuous"/>
    <DataField name="BOOKKEEPING_APPLICATION" optype="continuous"/>
    <DataField name="CUST_MARITAL_STATUS" optype="categorical"/>
    <DataField name="EDUCATION" optype="categorical"/>
    <DataField name="HOUSEHOLD_SIZE" optype="categorical"/>
    <DataField name="OCCUPATION" optype="categorical"/>
    <DataField name="YRS_RESIDENCE" optype="continuous"/>
    <DataField name="Y_BOX_GAMES" optype="continuous"/>
  </DataDictionary>
```

```
  <TreeModel modelName="DT_SH_CLAS_SAMPLE" functionName="classification"
      splitCharacteristic="binarySplit">
    <Extension name="buildSettings">
      <Setting name="TREE_IMPURITY_METRIC" value="TREE_IMPURITY_GINI"/>
      <Setting name="TREE_TERM_MAX_DEPTH" value="7"/>
      <Setting name="TREE_TERM_MINPCT_NODE" value=".05"/>
      <Setting name="TREE_TERM_MINPCT_SPLIT" value=".1"/>
      <Setting name="TREE_TERM_MINREC_NODE" value="10"/>
      <Setting name="TREE_TERM_MINREC_SPLIT" value="20"/>
      <costMatrix>
        <costElement>
          <actualValue>0</actualValue>
          <predictedValue>0</predictedValue>
          <cost>0</cost>
        </costElement>
        <costElement>
          <actualValue>0</actualValue>
          <predictedValue>1</predictedValue>
          <cost>1</cost>
        </costElement>
        <costElement>
          <actualValue>1</actualValue>
          <predictedValue>0</predictedValue>
          <cost>8</cost>
        </costElement>
        <costElement>
          <actualValue>1</actualValue>
          <predictedValue>1</predictedValue>
          <cost>0</cost>
        </costElement>
      </costMatrix>
    </Extension>
    <MiningSchema>
      .
      .
      .
      .
      .
      .
      </Node>
    </Node>
  </TreeModel>
</PMML>
```

GET_MODEL_TRANSFORMATIONS Function

This function returns the transformation expressions embedded in the specified model. Starting from Oracle Database 12*c* Release 2, this function is deprecated. See "Static Data Dictionary Views: `ALL_ALL_TABLES` to `ALL_OUTLINES`" in *Oracle Database Reference*.

All `GET_*` interfaces are replaced by model views, and Oracle recommends that users reference the model views to retrieve the relevant information. The `GET_MODEL_TRANSFORMATIONS` function is replaced by the following:

- USER(/DBA/ALL)_MINING_MODEL_XFORMS: provides the user-embedded transformations
- DM$VX prefixed model view: provides text feature extraction information
- D$VN prefixed mode view: provides normalization and missing value information
- DM$VB: provides binning information

> **See Also:**
>
> "About Transformation Lists" in DBMS_DATA_MINING_TRANSFORM Operational Notes
>
> "GET_TRANSFORM_LIST Procedure"
>
> "CREATE_MODEL Procedure"
>
> "ALL_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> "DBA_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> "USER_MINING_MODEL_XFORMS" in *Oracle Database Reference*
>
> Model Details View for Binning
>
> Normalization and Missing Value Handling
>
> Data Preparation for Text Features

Syntax

```
DBMS_DATA_MINING.get_model_transformations(
      model_name IN VARCHAR2,
      partition_name IN VARCHAR2 DEFAULT NULL)
  RETURN DM_Transforms PIPELINED;
```

Parameters

Table 62-113 GET_MODEL_TRANSFORMATIONS Function Parameters

Parameter	Description
`model_name`	Indicates the name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.
`partition_name`	Specifies a partition in a partitioned model

Return Values

Table 62-114 GET_MODEL_TRANSFORMATIONS Function Return Value

Return Value	Description

`DM_TRANSFORMS`	The transformation expressions embedded in *`model_name`*.
	The `DM_TRANSFORMS` type is a table of `DM_TRANSFORM` objects. Each `DM_TRANSFORM` has these fields:
	`attribute_name        VARCHAR2(4000)`
	`attribute_subname     VARCHAR2(4000)`
	`expression            CLOB`
	`reverse_expression    CLOB`

Usage Notes

When Automatic Data Preparation (ADP) is enabled, both automatic and user-defined transformations may be associated with an attribute. In this case, the user-defined transformations are evaluated before the automatic transformations.

When invoked for a partitioned model, the `partition_name` parameter must be specified.

Examples

In this example, several columns in the `SH.CUSTOMERS` table are used to create a naive Bayes model. A transformation expression is specified for one of the columns. The model does not use ADP.

```
CREATE OR REPLACE VIEW mining_data AS
   SELECT cust_id, cust_year_of_birth, cust_income_level,cust_credit_limit
   FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_INCOME_LEVEL                                  VARCHAR2(30)
 CUST_CREDIT_LIMIT                                  NUMBER

CREATE TABLE settings_nb(
      setting_name  VARCHAR2(30),
      setting_value VARCHAR2(30));
BEGIN
     INSERT INTO settings_nb (setting_name, setting_value) VALUES
           (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
     INSERT INTO settings_nb (setting_name, setting_value) VALUES
           (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_off);
     COMMIT;
END;
/
DECLARE
    mining_data_xforms   dbms_data_mining_transform.TRANSFORM_LIST;
  BEGIN
    dbms_data_mining_transform.SET_TRANSFORM (
         xform_list           =>  mining_data_xforms,
         attribute_name       => 'cust_year_of_birth',
         attribute_subname    =>  null,
         expression           => 'cust_year_of_birth + 10',
         reverse_expression   => 'cust_year_of_birth - 10');
    dbms_data_mining.CREATE_MODEL (
        model_name            =>  'new_model',
        mining_function       =>   dbms_data_mining.classification,
        data_table_name       =>  'mining_data',
        case_id_column_name   =>  'cust_id',
        target_column_name    =>  'cust_income_level',
        settings_table_name   =>  'settings_nb',
        data_schema_name      =>   nulL,
        settings_schema_name =>   null,
        xform_list            =>   mining_data_xforms );
  END;
 /
SELECT attribute_name, TO_CHAR(expression), TO_CHAR(reverse_expression)
      FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('new_model'));
```

```
ATTRIBUTE_NAME      TO_CHAR(EXPRESSION)       TO_CHAR(REVERSE_EXPRESSION)
------------------  ------------------------  -----------------------------
CUST_YEAR_OF_BIRTH  cust_year_of_birth + 10   cust_year_of_birth - 10
```

Related Topics

- *Oracle Database Reference*

GET_TRANSFORM_LIST Procedure

This procedure converts transformation expressions specified as `DM_TRANSFORMS` to a transformation list (`TRANSFORM_LIST`) that can be used in creating a model. `DM_TRANSFORMS` is returned by the `GET_MODEL_TRANSFORMATIONS` function.

You can also use routines in the `DBMS_DATA_MINING_TRANSFORM` package to construct a transformation list.

> **See Also:**
>
> "About Transformation Lists" in DBMS_DATA_MINING_TRANSFORM
>
> "GET_MODEL_TRANSFORMATIONS Function"
>
> "CREATE_MODEL Procedure"

Syntax

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

Parameters

Table 62-115 GET_TRANSFORM_LIST Procedure Parameters

Parameter	Description

`xform_list`	A list of transformation specifications that can be embedded in a model. Accepted as a parameter to the CREATE_MODEL Procedure.
	The `TRANSFORM_LIST` type is a table of `TRANSFORM_REC` objects. Each `TRANSFORM_REC` has these fields:
	`attribute_name      VARCHAR2(30)`
	`attribute_subname   VARCHAR2(4000)`
	`expression          EXPRESSION_REC`
	`reverse_expression  EXPRESSION_REC`
	`attribute_spec      VARCHAR2(4000)`
	For details about the `TRANSFORM_LIST` collection type, see Table 63-1.

Table 62-115 (Cont.) GET_TRANSFORM_LIST Procedure Parameters

Parameter	Description

`model_xforms`	A list of embedded transformation expressions returned by the GET_MODEL_TRANSFORMATIONS Function for a specific model.
	The `DM_TRANSFORMS` type is a table of `DM_TRANSFORM` objects. Each `DM_TRANSFORM` has these fields:
	`attribute_name VARCHAR2(4000)`
	`attribute_subname VARCHAR2(4000)`
	`expression CLOB`
	`reverse_expression CLOB`

Examples

In this example, a model `mod1` is trained using several columns in the `SH.CUSTOMERS` table. The model uses ADP, which automatically bins one of the columns.

A second model `mod2` is trained on the same data without ADP, but it uses a transformation list that was obtained from `mod1`. As a result, both `mod1` and `mod2` have the same embedded transformation expression.

```
CREATE OR REPLACE VIEW mining_data AS
     SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit
     FROM sh.customers;

describe mining_data
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUST_ID                                   NOT NULL NUMBER
 CUST_YEAR_OF_BIRTH                        NOT NULL NUMBER(4)
 CUST_INCOME_LEVEL                                  VARCHAR2(30)
 CUST_CREDIT_LIMIT                                  NUMBER

CREATE TABLE setmod1(setting_name  VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
   INSERT INTO setmod1 VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
   INSERT INTO setmod1 VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
   dbms_data_mining.CREATE_MODEL (
               model_name            => 'mod1',
               mining_function       => dbms_data_mining.classification,
               data_table_name       => 'mining_data',
               case_id_column_name   => 'cust_id',
               target_column_name    => 'cust_income_level',
               settings_table_name   => 'setmod1');
    COMMIT;
END;
/
CREATE TABLE setmod2(setting_name  VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
  INSERT INTO setmod2
      VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
  COMMIT;
END;
/
DECLARE
  v_xform_list       dbms_data_mining_transform.TRANSFORM_LIST;
  dmxf               DM_TRANSFORMS;
```

```
BEGIN
   EXECUTE IMMEDIATE
    'SELECT dm_transform(attribute_name, attribute_subname,expression, reverse_expression)
     FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS (''mod1''))'
     BULK COLLECT INTO dmxf;
   dbms_data_mining.GET_TRANSFORM_LIST (
        xform_list             =>  v_xform_list,
        model_xforms           =>  dmxf);
   dbms_data_mining.CREATE_MODEL(
         model_name            => 'mod2',
         mining_function       =>  dbms_data_mining.classification,
         data_table_name       => 'mining_data',
         case_id_column_name   => 'cust_id',
         target_column_name    => 'cust_income_level',
         settings_table_name   => 'setmod2',
         xform_list            =>  v_xform_list);
END;
/

-- Transformation expression embedded in mod1
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(EXPRESSION)
--------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Transformation expression embedded in mod2
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(EXPRESSION)
--------------------------------------------------------------------------------
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

-- Reverse transformation expression embedded in mod1
SELECT TO_CHAR(reverse_expression)FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5;  )',NULL,'NULL')

-- Reverse transformation expression embedded in mod2
SELECT TO_CHAR(reverse_expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));

TO_CHAR(REVERSE_EXPRESSION)
--------------------------------------------------------------------------------
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915), [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
```

```
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5;  )',NULL,'NULL')
```

IMPORT_MODEL Procedure

This procedure imports one or more machine learning models. The procedure is overloaded. You can call it to import machine learning models from a dump file set, or you can call it to import a single machine learning model from a PMML document.

Import from a dump file set

You can import machine learning models from a dump file set that was created by the EXPORT_MODEL Procedure. `IMPORT_MODEL` and `EXPORT_MODEL` use Oracle Data Pump technology to export to and import from a dump file set.

When Oracle Data Pump is used directly to export/import an entire schema or database, the machine learning models in the schema or database are included. `EXPORT_MODEL` and `IMPORT_MODEL` export/import machine learning models only.

Import from PMML

You can import a machine learning model represented in Predictive Model Markup Language (PMML). The model must be of type `RegressionModel`, either linear regression or binary logistic regression.

PMML is an XML-based standard specified by the Data Mining Group (`https://dmg.org`). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Machine Learning for SQL supports the core features of PMML 3.1 for regression models.

See Also:

Oracle Machine Learning for SQL User's Guide for more information about exporting and importing machine learning models

Oracle Database Utilities for information about Oracle Data Pump

`https://dmg.org/dmg-faq.html` for more information about PMML

Syntax

Imports a machine learning model from a dump file set:

```
DBMS_DATA_MINING.IMPORT_MODEL (
      filename          IN  VARCHAR2,
      directory         IN  VARCHAR2,
      model_filter      IN  VARCHAR2 DEFAULT NULL,
      operation         IN  VARCHAR2 DEFAULT NULL,
      remote_link       IN  VARCHAR2 DEFAULT NULL,
      jobname           IN  VARCHAR2 DEFAULT NULL,
      schema_remap      IN  VARCHAR2 DEFAULT NULL,
      tablespace_remap  IN  VARCHAR2 DEFAULT NULL);
```

Imports a machine learning model from a PMML document:

```
DBMS_DATA_MINING.IMPORT_MODEL (
      model_name         IN  VARCHAR2,
      pmmldoc            IN  XMLTYPE
      strict_check       IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 62-116 IMPORT_MODEL Procedure Parameters

Parameter	Description
`filename`	Name of the dump file set from which the models should be imported. The dump file set must have been created by the `EXPORT_MODEL` procedure or the `expdp` export utility of Oracle Data Pump. The dump file set can contain one or more files. (Refer to "EXPORT_MODEL Procedure" for details.) If the dump file set contains multiple files, you can specify `'`*`filename`*`%U'` instead of listing them. For example, if your dump file set contains 3 files, `archive01.dmp`, `archive02.dmp`, and `archive03.dmp`, you can import them by specifying `'archive%U'`.
`directory`	Name of a pre-defined directory object that specifies where the dump file set is located. Both the exporting and the importing user must have read/write access to the directory object and to the file system directory that it identifies. Note: The target database must have also have read/write access to the file system directory.

`model_filter`	Optional parameter that specifies one or more models to import. If you do not specify a value for `model_filter`, all models in the dump file set are imported. You can also specify `NULL` (the default) or 'ALL' to import all models.
	The value of `model_filter` can be one or more model names. The following are valid filters.
	`'mymodel1'`
	`'name IN (''mymodel2'',''mymodel3'')'`
	The first causes `IMPORT_MODEL` to import a single model named `mymodel1`. The second causes `IMPORT_MODEL` to import two models, `mymodel2` and `mymodel3`.

`operation`	Optional parameter that specifies whether to import the models or the SQL statements that create the models. By default, the models are imported. You can specify either of the following values for `operation`: • `'IMPORT'` — Import the models (Default) • `'SQL_FILE'`— Write the SQL DDL for creating the models to a text file. The text file is named *`job_name`*`.sql` and is located in the dump set directory.
`remote_link`	Optional parameter that specifies the name of a database link to a remote system. The default value is `NULL`. A database link is a schema object in a local database that enables access to objects in a remote database. When you specify a value for `remote_link`, you can import models into the local database from the remote database. The import is fileless; no dump file is involved. The `IMP_FULL_DATABASE` role is required for importing the remote models. The `EXP_FULL_DATABASE` privilege, the `CREATE DATABASE LINK` privilege, and other privileges may also be required. (See Example 2.)
`jobname`	Optional parameter that specifies the name of the import job. By default, the name has the form *`username`*`_imp_`*`nnnn`*, where *`nnnn`* is a number. For example, a job name in the `SCOTT` schema might be `SCOTT_imp_134`. If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters. A log file for the import job, named *`jobname`*`.log`, is created in the same directory as the dump file set.

Table 62-116 (Cont.) IMPORT_MODEL Procedure Parameters

Parameter	Description
`schema_remap`	Optional parameter for importing into a different schema. By default, models are exported and imported within the same schema. If the dump file set belongs to a different schema, you must specify a schema mapping in the form *`export_user`*:*`import_user`*. For example, you would specify `'SCOTT:MARY'` to import a model exported by `SCOTT` into the `MARY` schema. Note: In some cases, you may need to have the `IMP_FULL_DATABASE` privilege or the `SYS` role to import a model from a different schema.
`tablespace_remap`	Optional parameter for importing into a different tablespace. By default, models are exported and imported within the same tablespace. If the dump file set belongs to a different tablespace, you must specify a tablespace mapping in the form *`export_tablespace`*:*`import_tablespace`*. For example, you would specify `'TBLSPC01:TBLSPC02'` to import a model that was exported from tablespace `TBLSPC01` into tablespace `TBLSPC02`. Note: In some cases, you may need to have the `IMP_FULL_DATABASE` privilege or the `SYS` role to import a model from a different tablespace.
`model_name`	Name for the new model that will be created in the database as a result of an import from PMML The name must be unique within the user's schema.
`pmmldoc`	The PMML document representing the model to be imported. The PMML document has an `XMLTYPE` object type. See "XMLTYPE" for details.
`strict_check`	Whether or not an error occurs when the PMML document contains sections that are not part of core PMML (for example, Output or Targets). OML4SQL supports only core PMML; any non-core features may affect the scoring representation. If the PMML does not strictly conform to core PMML and `strict_check` is set to `TRUE`, then `IMPORT_MODEL` returns an error. If strict_check is `FALSE` (the default), then the error is suppressed. The model may be imported and scored.

Examples

1. This example shows a model being exported and imported within the schema `oml_user2`. Then the same model is imported into the `oml_user3` schema. The `oml_user3` user has the `IMP_FULL_DATABASE` privilege. The `oml_user2` user has been assigned the `USER2` tablespace; `oml_user3` has been assigned the `USER3` tablespace.

```
SQL> connect oml_user2
Enter password: oml_user2_password
Connected.
SQL> select model_name from user_mining_models;

MODEL_NAME
------------------------------
NMF_SH_SAMPLE
SVMO_SH_CLAS_SAMPLE
SVMR_SH_REGR_SAMPLE

-- export the model called NMF_SH_SAMPLE to a dump file in same schema
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL (
            filename =>'NMF_SH_SAMPLE_out',
            directory =>'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''');

-- import the model back into the same schema
```

```
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (
            filename => 'NMF_SH_SAMPLE_out01.dmp',
            directory => 'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''');

-- connect as different user
-- import same model into that schema
SQL> connect oml_user3
Enter password: oml_user3_password
Connected.
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL (
            filename => 'NMF_SH_SAMPLE_out01.dmp',
            directory => 'DATA_PUMP_DIR',
            model_filter => 'name = ''NMF_SH_SAMPLE''',
            operation =>'IMPORT',
            remote_link => NULL,
            jobname => 'nmf_imp_job',
            schema_remap => 'oml_user2:oml_user3',
            tablespace_remap => 'USER2:USER3');
```

The following example shows user `MARY` importing all models from a dump file, `model_exp_001.dmp`, which was created by user `SCOTT`. User `MARY` has been assigned a tablespace named `USER2`; user `SCOTT` was assigned the tablespace `USERS` when the models were exported into the dump file `model_exp_001.dmp`.The dump file is located in the file system directory mapped to a directory object called `DM_DUMP`. If user `MARY` does not have `IMP_FULL_DATABASE` privileges, `IMPORT_MODEL` will raise an error.

```
-- import all models
DECLARE
  file_name  VARCHAR2(40);
BEGIN
  file_name := 'model_exp_001.dmp';
  DBMS_DATA_MINING.IMPORT_MODEL(
            filename=> 'file_name',
            directory=>'DM_DUMP',
            schema_remap=>'SCOTT:MARY',
            tablespace_remap=>'USERS:USER2');
  DBMS_OUTPUT.PUT_LINE(
           'DBMS_DATA_MINING.IMPORT_MODEL of all models from SCOTT done!');
END;
/
```

2. This example shows how the user `xuser` could import the model `oml_user.r1mod` from a remote database. The SQL*Net connection alias for the remote database is `R1DB`. The user `xuser` is assigned the `SYSAUX` tablespace; the user `oml_user` is assigned the `TBS_1` tablespace.

```
CONNECT / AS SYSDBA;
GRANT CREATE DATABASE LINK TO xuser;
GRANT imp_full_database TO xuser;
CONNECT xuser/xuserpassword
CREATE DATABASE LINK oml_user_link
         CONNECT TO oml_user IDENTIFIED BY oml_userpassword USING 'R1DB';
EXEC dbms_data_mining.import_model (
    NULL,
   'oml_user_DIR',
   'R1MOD',
    remote_link => 'oml_user_LINK', schema_remap => 'oml_user:XUSER',
                    tablespace_remap => 'TBS_1:SYSAUX' );
SELECT name FROM dm_user_models;

NAME
```

```
-----------------------------------------------------------------------------
R1MOD
```

3. This example shows how a PMML document called `SamplePMML1.xml` could be imported from a location referenced by directory object `PMMLDIR` into the schema of the current user. The imported model will be called `PMMLMODEL1`.

```
BEGIN
    dbms_data_mining.import_model ('PMMLMODEL1',
        XMLType (bfilename ('PMMLDIR', 'SamplePMML1.xml'),
          nls_charset_id ('AL32UTF8')
        ));
END;
```

IMPORT_ONNX_MODEL Procedure

This procedure enables you to import an ONNX model into the Database.

Syntax

```
DBMS_DATA_MINING.IMPORT_ONNX_MODEL(
model_name   IN  VARCHAR2,
model_data   IN  BLOB,
metadata     IN  JSON);
```

Parameters

Table 62-117 IMPORT_ONNX_MODEL Procedure Parameters

Parameter	Description
`model_name`	Name of the model in the form `[schema_name.]model_name`. If you do not specify a schema, then your own schema is used.
`model_data`	It is a `BLOB` holding the ONNX representation of the model. The `BLOB` contains the identical byte sequence as the one stored in an ONNX file.
`metadata`	A JSON description of the metadata describing the model. The metadata at minimum must describe the machine learning function supported by the model. The model's metadata parameters are described in JSON Metadata Parameters for ONNX Models .

Example

The following example illustrates a code snippet of using the `DBMS_DATA_MINING.IMPORT_ONNX_MODEL` procedure. The complete step-by-step example is illustrated in Import ONNX Models and Generate Embeddings and Alternate Method to Import ONNX Models.

```
DBMS_DATA_MINING.IMPORT_ONNX_MODEL('my_embedding_model.onnx',
                                            :blob_bind_variable,
                                             JSON('{"function" :
"embedding",
                                                           "embeddingOutput" :
"embedding" ,
                                                            "input":{"input":
["DATA"]}}'));
```

For a complete example to illustrate how you can define a `BLOB` variable and use it in the `IMPORT_ONNX_MODEL` procedure, you can have the following:

```
CREATE OR REPLACE MY_LOAD_EMBEDDING_MODEL(embedding_model_name VARCHAR2,
onnx_blob BLOB) IS
BEGIN
DBMS_DATA_MINING.IMPORT_ONNX_MODEL(embedding_model_name,
                            onnx_blob,
                            JSON('{"function" : "embedding",
                                   "embeddingOutput" : "embedding" ,
                                   "input":{"input": ["DATA"]}}'));
END;
/
```

Usage Notes

The name of the model follows the same restrictions as those used for other machine learning models, namely:

- The schema name, if provided, is limited to 128 characters.
- The model name is limited to 123 characters and must follow the rules of unquoted identifiers: they contain only alphanumeric characters, the underscore (_), dollar sign ($), and pound sign (#). The initial character must be alphabetic.
- The model size is limited to 1 gigabyte.
- The model must not depend on external initializers. To know more about initializers and other ONNX concepts, see https://onnx.ai/onnx/intro/concepts.html.

IMPORT_SERMODEL Procedure

This procedure imports the serialized format of the model back into a database.

The import routine takes the serialized content in the `BLOB` and the name of the model to be created with the content. This import does not create model views or tables that are needed for querying model details. The import procedure only provides the ability to score the model.

Syntax

```
DBMS_DATA_MINING.IMPORT_SERMODEL (
      model_data      IN BLOB,
      model_name      IN VARCHAR2,);
```

Parameters

Table 62-118 IMPORT_SERMODEL Procedure Parameters

Parameter	Description
`model_data`	Provides model data in `BLOB` format.
`model_name`	Name of the machine learning model in the form [*schema_name*.]*model_name*. If you do not specify a schema, then your own schema is used.

Examples

The following statement imports the serialized format of the models.

```
declare
 v_blob blob;
BEGIN
 dbms_lob.createtemporary(v_blob, FALSE);
-- fill in v_blob from somewhere (e.g., bfile, etc.)
 dbms_data_mining.import_sermodel(v_blob, 'MY_MODEL');
 dbms_lob.freetemporary(v_blob);
END;
/
```

Related Topics

- EXPORT_SERMODEL Procedure
  This procedure exports the model in a serialized format so that they can be moved to another platform for scoring.

> **See Also:**
>
> *Oracle Machine Learning for SQL User's Guide* for more information about exporting and importing machine learning models

JSON Schema for R Extensible Algorithm

Provides some flexibility when creating a new JSON object following the JSON schema.

Usage Note

Some flexibility when creating a new JSON object is as follows:

- Partial registration is allowed. For example, the detail function can be missing.
- Different orders are allowed. For example, the detail function can be written before the build function or after it.

Example 62-1 JSON Schema

JSON schema 1.1 for R extensible algorithm:

```
{
    "type": "object",
    "properties": {
        "algo_name_display": { "type" : "object",
                                        "properties" : {
                                        "language" : { "type" :
"string",

"enum" : ["English", "Spanish", "French"],

"default" : "English"},
```

```
                                        "name" : { "type" : "string"}}
                                      },

            "function_language": {"type": "string" },
            "mining_function": {
                     "type" : "array",
                     "items" : [
                         { "type" : "object",
                            "properties" : {
                               "mining_function_name"  : { "type" : "string"},
                               "build_function": {
                                       "type": "object",
                                       "properties": {
                                            "function_body": { "type": "CLOB" }
                                                            }
                                        },

            "detail_function": {
                     "type" : "array",
                      "items" : [
                          {"type": "object",
                            "properties": {
                                 "function_body": { "type": "CLOB" },
                                 "view_columns": { "type" : "array",
                                                                    "items" : {

    "type" : "object",

    "properties" : {

     "name" : { "type" : "string"},

     "type" : { "type" : "string",

                     "enum" : ["VARCHAR2",

                                      "NUMBER",

                                      "DATE",

                                      "BOOLEAN"]

                   }
                                                                                     }
                                                                   }
                                                      }
                                          }
                               ]
            },

           "score_function": {
                     "type": "object",
                     "properties": {
                           "function_body": { "type": "CLOB" }
                            }
                     },
```

```
          "weight_function": {
                         "type": "object",
                         "properties": {
                             "function_body": { "type": "CLOB" },
                         }
                  }
                                }
             }]
          },

         "algo_setting": {
                  "type" : "array",
                  "items" : [
                      { "type" : "object",
                         "properties" : {
                            "name"              : { "type" : "string"},
                            "name_display": { "type" : "object",
                                                           "properties" : {
                                                           "language" :
{ "type" : "string",

   "enum" : ["English", "Spanish", "French"],

   "default" : "English"},
                                                           "name" : { "type" :
"string"}}
                                                        },
                            "type" : { "type" : "string",
                                            "enum" : ["string", "integer",
"number", "boolean"]},

                            "optional": {"type" : "BOOLEAN",
                                                 "default" : "FALSE"},

                            "value" : { "type" :  "string"},

                            "min_value" : { "type": "object",
                                                        "properties": {
                                                              "min_value":
{"type": "number"},
                                                               "inclusive":
{ "type": "boolean",

      "default" : TRUE},
                                                         }
                                                   },
                             "max_value" : {"type": "object",
                                                      "properties": {
                                                           "max_value":
{"type": "number"},
                                                           "inclusive":
{ "type": "boolean",

   "default" : TRUE},
                                                                }
                                                      },
```

```
                          "categorical choices" : { "type": "array",
                                                          "items": {
                                                              "type":
"string"
                                                            }
                                                        },

                          "description_display": { "type" : "object",

"properties" : {

"language" : { "type" : "string",

            "enum" : ["English", "Spanish", "French"],

            "default" : "English"},
                                                          "name" :
{ "type" : "string"}}
                                                        }
                        }
                    }
                ]
            }
      }
}
```

Example 62-2 JSON object example

The following is an JSON object example that must be passed to the registration procedure:

```
{  "algo_name_display"   :     {"English", "t1"},
                         "function_language"    :      "R",
                         "mining_function" : {
  "mining_function_name" : "CLASSIFICATION",
                         "build_function" : {"function_body": "function(dat,
formula, family)
{
                                                         set.seed(1234);
                                          mod <- glm(formula = formula,
data=dat,
                                                      family=
eval(parse(text=family))); mod}"},
           "score_function" :  { "function_body": "function(mod, dat) {
                                             res <- predict(mod, newdata =
dat,
type=''response
                                         '');
                                             res2=data.frame(1-res, res);
res2}"}}
                          },
                          "algo_setting" :   [{"name"                :
"dbms_data_mining.odms_m


                                                      issing_value_treatment",
```

```
                            "name_display"   : {"English",
"dbms_data_mining.odms_missing_value
_treatment"},
                            "type"                  : "string",
                            "optional"            :  "TRUE",
                            "value"                :
"dbms_data_mining.odms_missing_value_mean_mode",
                            "categorical choices"   :
[    "dbms_data_mining.odms_missing_value_mean_mode",

"dbms_data_mining.odms_missing_value_auto",

"dbms_data_mining.odms_missing_value_delete_row"],
                            "description"                : {"English",
                                                                         "how to
treat missing values"}
                         },

{"name"                  : "RALG_PARAMETER_FAMILY",
                            "name_display"   : {"English",
"RALG_PARAMETER_FAMILY"},
                            "type"                  : "string",
                            "optional"            :  "TRUE",
                            "value"                :  "",
                            "description"       : {"English", "R family
parameter in build function"}
                         }
],
                        }
```

REGISTER_ALGORITHM Procedure

Use this function to register a new algorithm by providing the algorithm name, machine learning function, and all other algorithm metadata.

Syntax

```
DBMS_DATA_MINING.REGISTER_ALGORITHM (
                  algorithm_name            IN VARCHAR2,
                  algorithm_metadata        IN CLOB,
                  algorithm_description     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-119 ***REGISTER_ALGORITHM Procedure Parameters***

Parameter	Description
`algorithm_name`	Name of the algorithm.
`algorithm_metadata`	Metadata of the algorithm.
`algorithm_description`	Description of the algorithm.

Usage Notes

The registration procedure performs the following:

- Checks whether `algorithm_metadata` has correct JSON syntax.
- Checks whether the input JSON object follows the predefined JSON schema.
- Checks whether current user has `RQADMIN` privilege.
- Checks duplicate algorithms so that the same algorithm is not registered twice.
- Checks for missing entries. For example, algorithm name, algorithm type, metadata, and build function.

Register Algorithms After the JSON Object Is Created

SQL users can register new algorithms by creating a JSON object following the JSON schema and passing it to the `REGISTER_ALGORITHM` procedure.

```
BEGIN
  DBMS_DATA_MINING.register_algorithm(
    algorithm_name                 =>   't1',
    algorithm_metadata            =>
    '{"function_language" : "R",
      "mining_function" :
        { "mining_function_name" : "CLASSIFICATION",
           "build_function" : {"function_body": "function(dat, formula,
family) { set.seed(1234);
                                          mod <- glm(formula = formula,
data=dat,

family=eval(parse(text=family)));
mod}"},
           "score_function" :  {"function_body": "function(mod, dat) {
                                             res <- predict(mod, newdata =
dat, type=''response'');
                                             res2=data.frame(1-res, res);
res2}"}}
    }',
    algorithm_description  => 't1');
END;
/
```

RANK_APPLY Procedure

This procedure ranks the results of an `APPLY` operation based on a top-N specification for predictive and descriptive model results.

For classification models, you can provide a cost matrix as input, and obtain the ranked results with costs applied to the predictions.

Syntax

```
DBMS_DATA_MINING.RANK_APPLY (
      apply_result_table_name        IN VARCHAR2,
      case_id_column_name            IN VARCHAR2,
      score_column_name              IN VARCHAR2,
      score_criterion_column_name    IN VARCHAR2,
      ranked_apply_table_name        IN VARCHAR2,
      top_N                          IN NUMBER (38) DEFAULT 1,
      cost_matrix_table_name         IN VARCHAR2    DEFAULT NULL,
```

```
      apply_result_schema_name        IN VARCHAR2     DEFAULT NULL,
      cost_matrix_schema_name         IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table 62-120 RANK_APPLY Procedure Parameters

Parameter	Description
apply_result_table_name	Name of the table or view containing the results of an APPLY operation on the test data set (see Usage Notes)
case_id_column_name	Name of the case identifier column. This must be the same as the one used for generating APPLY results.
score_column_name	Name of the prediction column in the apply results table
score_criterion_column_name	Name of the probability column in the apply results table
ranked_apply_result_tab_name	Name of the table containing the ranked apply results
top_N	Top N predictions to be considered from the APPLY results for precision recall computation
cost_matrix_table_name	Name of the cost matrix table
apply_result_schema_name	Name of the schema hosting the APPLY results table
cost_matrix_schema_name	Name of the schema hosting the cost matrix table

Usage Notes

You can use RANK_APPLY to generate ranked apply results, based on a top-N filter and also with application of cost for predictions, if the model was built with costs.

The behavior of RANK_APPLY is similar to that of APPLY with respect to other DDL-like operations such as CREATE_MODEL, DROP_MODEL, and RENAME_MODEL. The procedure does not depend on the model; the only input of relevance is the apply results generated in a fixed schema table from APPLY.

The main intended use of RANK_APPLY is for the generation of the final APPLY results against the scoring data in a production setting. You can apply the model against test data using APPLY, compute various test metrics against various cost matrix tables, and use the candidate cost matrix for RANK_APPLY.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The case_id column will be the same case identifier column as that of the apply results.

Classification Models — NB and SVM

For numerical targets, the ranked results table will have the definition as shown:

```
(case_id        VARCHAR2/NUMBER,
prediction      NUMBER,
probability     NUMBER,
cost            NUMBER,
rank            INTEGER)
```

For categorical targets, the ranked results table will have the following definition:

```
(case_id        VARCHAR2/NUMBER,
prediction      VARCHAR2,
probability     NUMBER,
cost            NUMBER,
rank            INTEGER)
```

Clustering Using *k*-Means or O-Cluster

Clustering is an unsupervised machine learning function, and hence there are no targets. The results of an `APPLY` operation contains simply the cluster identifier corresponding to a case, and the associated probability. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the cluster ids ranked by `top-N`.

```
(case_id        VARCHAR2/NUMBER,
cluster_id      NUMBER,
probability     NUMBER,
rank            INTEGER)
```

Feature Extraction using NMF

Feature extraction is also an unsupervised machine learning function, and hence there are no targets. The results of an `APPLY` operation contains simply the feature identifier corresponding to a case, and the associated match quality. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the feature ids ranked by `top-N`.

```
(case_id         VARCHAR2/NUMBER,
feature_id       NUMBER,
match_quality    NUMBER,
rank             INTEGER)
```

Examples

```
BEGIN
/* build a model with name census_model.
 * (See example under CREATE_MODEL)
 */

/* if training data was pre-processed in any manner,
 * perform the same pre-processing steps on apply
 * data also.
 * (See examples in the section on DBMS_DATA_MINING_TRANSFORM)
 */

/* apply the model to data to be scored */
DBMS_DATA_MINING.RANK_APPLY(
  apply_result_table_name       => 'census_apply_result',
  case_id_column_name           => 'person_id',
  score_column_name             => 'prediction',
  score_criterion_column_name   => 'probability
  ranked_apply_result_tab_name  => 'census_ranked_apply_result',
  top_N                         => 3,
  cost_matrix_table_name        => 'census_cost_matrix');
END;
/

-- View Ranked Apply Results
SELECT *
  FROM census_ranked_apply_result;
```

REMOVE_COST_MATRIX Procedure

The REMOVE_COST_MATRIX procedure removes the default scoring matrix from a classification model.

> **See Also:**
>
> - "ADD_COST_MATRIX Procedure"
> - "REMOVE_COST_MATRIX Procedure"

Syntax

```
DBMS_DATA_MINING.REMOVE_COST_MATRIX (
      model_name    IN   VARCHAR2);
```

Parameters

Table 62-121 Remove_Cost_Matrix Procedure Parameters

Parameter	Description
model_name	Name of the model in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used.

Usage Notes

If the model is not in your schema, then REMOVE_COST_MATRIX requires the ALTER ANY MINING MODEL system privilege or the ALTER object privilege for the machine learning model.

Example

The naive Bayes model NB_SH_CLAS_SAMPLE has an associated cost matrix that can be used for scoring the model.

```
SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;

ACTUAL     PREDICTED        COST
---------- ---------- ----------
0          0                   0
1          0                 .75
0          1                 .25
1          1                   0
```

You can remove the cost matrix with REMOVE_COST_MATRIX.

```
SQL>EXECUTE dbms_data_mining.remove_cost_matrix('nb_sh_clas_sample');

SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;

no rows selected
```

RENAME_MODEL Procedure

This procedure changes the name of the machine learning model indicated by *model_name* to the name that you specify as *new_model_name*.

If a model with *new_model_name* already exists, then the procedure optionally renames *new_model_name* to *versioned_model_name* before renaming *model_name* to *new_model_name*.

The model name is in the form [*schema_name*.]*model_name*. If you do not specify a schema, your own schema is used. For machine learning model naming restrictions, see the Usage Notes for "CREATE_MODEL Procedure".

Syntax

```
DBMS_DATA_MINING.RENAME_MODEL (
     model_name             IN VARCHAR2,
     new_model_name         IN VARCHAR2,
     versioned_model_name   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 62-122 RENAME_MODEL Procedure Parameters

Parameter	Description
`model_name`	Model to be renamed.
`new_model_name`	New name for the model *`model_name`*.
`versioned_model_name`	New name for the model *`new_model_name`* if it already exists.

Usage Notes

If you attempt to rename a model while it is being applied, then the model will be renamed but the apply operation will return indeterminate results.

Examples

1. This example changes the name of model `census_model` to `census_model_2012`.

   ```
   BEGIN
     DBMS_DATA_MINING.RENAME_MODEL(
       model_name      => 'census_model',
       new_model_name  => 'census_model_2012');
   END;
   /
   ```

2. In this example, there are two classification models in the user's schema: `clas_mod`, the working model, and `clas_mod_tst`, a test model. The `RENAME_MODEL` procedure preserves `clas_mod` as `clas_mod_old` and makes the test model the new working model.

   ```
   SELECT model_name FROM user_mining_models;
   MODEL_NAME
   -------------------------------------------------------------------
   CLAS_MOD
   CLAS_MOD_TST

   BEGIN
     DBMS_DATA_MINING.RENAME_MODEL(
   ```

```
    model_name            => 'clas_mod_tst',
    new_model_name        => 'clas_mod',
    versioned_model_name  => 'clas_mod_old');
END;
/

SELECT model_name FROM user_mining_models;
MODEL_NAME
--------------------------------------------------------------------
CLAS_MOD
CLAS_MOD_OLD
```