Oracle® Data Mining

Application Developer's Guide

10*g* Release 1 (10.1)

Part No. B10699-01

December 2003

ORACLE®

Oracle Data Mining Application Developer's Guide, 10*g* Release 1 (10.1).

Part No. B10699-01

Primary Authors: Gina Abeles, Ramkumar Krishnan, Mark Hornick, Denis Mukhin, George Tang, Shiby Thomas, Sunil Venkayala.

Contributors: Marcos Campos, James McEvoy, Boriana Milenova, Margaret Taft, Joseph Yarmus.

Contents

3 ODM Java API Basic Usage

4 DBMS_DATA_MINING

Index

Send Us Your Comments

Oracle Data Mining Application Developer's Guide, 10*g* Release 1 (10.1)

Part No. B10699-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: 781-238-9893 Attn: Oracle Data Mining Documentation
- Postal service:
  Oracle Corporation
  Oracle Data Mining Documentation
  10 Van de Graaff Drive
  Burlington, Massachusetts 01803
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes using the Oracle Data Mining Java and PL/SQL Application Programming Interfaces (APIs) to perform data mining tasks for business applications, bioinformatics, and text mining.

Intended Audience

This manual is intended for anyone planning to write programs using the Oracle Data Mining Java or PL/SQL interface.

Familiarity with Java or PL/SQL is assumed, as well as familiarity with databases and data mining.

Users of the Oracle Data Mining BLAST table functions should be familiar with NCBI BLAST and related concepts.

Structure

This manual is organized as follows:

- Chapter 1, "Introduction"
- Chapter 2, "ODM Java Programming"
- Chapter 3, "ODM Java API Basic Usage"
- Chapter 4, "DBMS_DATA_MINING"
- Chapter 5, "ODM PL/SQL Sample Programs"
- Chapter 6, "Sequence Matching and Annotation (BLAST)"
- Chapter 7, "Text Mining"

- Appendix A, "Binning"
- Appendix B, "ODM Tips and Techniques"

Where to Find More Information

The documentation set for Oracle Data Mining is part of the Oracle 10*g* Database Documentation Library. The ODM documentation set consists of the following documents, available online:

- *Oracle Data Mining Administrator's Guide*, Release 10*g*
- *Oracle Data Mining Concepts*, Release 10*g*
- *Oracle Data Mining Application Developer's Guide*, Release 10*g* (this document)

Last-minute information about ODM is provided in the platform-specific README file.

For detailed information about the Java API, see the ODM Javadoc in the directory `$ORACLE_HOME/dm/doc/jdoc` (UNIX) or `%ORACLE_HOME%\dm\doc\jdoc` (Windows) on any system where ODM is installed.

For detailed information about the PL/SQL interface, see the *Supplied PL/SQL Packages and Types Reference*.

For information about the data mining process in general, independent of both industry and tool, a good source is the CRISP-DM project (Cross-Industry Standard Process for Data Mining) (http://www.crisp-dm.org/).

Related Manuals

For more information about the database underlying Oracle Data Mining, see:

- *Oracle Administrator's Guide*, Release 10*g*
- *Oracle Database 10g Installation Guide* for your platform.

For information about developing applications to interact with the Oracle Database, see

- *Oracle Application Developer's Guide — Fundamentals*, Release 10g

For information about upgrading from Oracle Data Mining release 9.0.1 or release 9.2.0, see

- *Oracle Database Upgrade Guide*, Release 10*g*
- *Oracle Data Mining Administrator's Guide*, Release 10*g*

For information about installing Oracle Data Mining, see

- *Oracle Installation Guide*, Release 10*g*
- *Oracle Data Mining Administrator's Guide*, Release 10*g*

Conventions

In this manual, Windows refers to the Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP operating systems.

The SQL interface to Oracle is referred to as SQL. This interface is the Oracle implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also followed in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface	Boldface type in text indicates the name of a class or method.
italic text	Italic type in text indicates a term defined in the text, the glossary, or in both locations.
`typewriter`	In interactive examples, user input is indicated by bold typewriter font, and system output by plain typewriter font.
`typewriter`	Terms in italic typewriter font represent placeholders or variables.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none

Documentation Accessibility

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

1
Introduction

Oracle Data Mining embeds data mining in the Oracle database. The data never leaves the database — the data, data preparation, model building, and model scoring activities all remain in the database. This enables Oracle to provide an infrastructure for data analysts and application developers to integrate data mining seamlessly with database applications.

Oracle Data Mining is designed for programmers, systems analysts, project managers, and others interested in developing database applications that use data mining to discover hidden patterns and use that knowledge to make predictions.

There are two interfaces: a Java API and a PL/SQL API. The Java API assumes a working knowledge of Java, and the PL/SQL API assumes a working knowledge of PL/SQL. Both interfaces assume a working knowledge of application programming and familiarity with SQL to access information in relational database systems.

This document describes using the Java and PL/SQL interface to write application programs that use data mining. It is organized as follows:

- Chapter 1 introduces ODM.
- Chapter 2 and Chapter 3 describe the Java interface. Chapter 2 provides an overview; Chapter 3 provides details. Reference information for methods and classes is available with Javadoc. The demo Java programs are described in Table 3–1. The demo programs are available as part of the installation; see the README file for details.
- Chapter 4 and Chapter 5 describe the PL/SQL interface. Basics are described inChapter 4, and demo PL/SQL programs are described in Chapter 5.
- Reference information for the PL/SQL functions and procedures is included in the *PL/SQL Packages and Types Reference.* The demo programs themselves are available as part of the installation; see the README file for details.

- Chapter 6 describes programming with BLAST, a set of table functions for performing sequence matching searches against nucleotide and amino acid sequence data stored in an Oracle database.
- Chapter 7 describes how to use the PL/SQL interface to do text mining.
- Appendix A contains an example of binning.
- Appendix B provides tips and techniques useful in both the Java and the PL/SQL interface.

1.1 ODM Requirements and Constraints

Anyone writing an Oracle Data Mining program must observe the following requirements and constraints:

- **Attribute Names in ODM:** All attribute names in ODM are case-sensitive and limited to 30 bytes in length; that is, attribute names may be quoted strings that contain mixed-case characters and/or special characters. Simply put, attribute names used by ODM follow the same naming conventions and restrictions as column names or type attribute names in Oracle.
- **Mining Object Names in ODM:** All mining object names in ODM are 25 or fewer bytes in length and must be uppercase only. Model names may contain the underscore ("_") but no other special characters. Certain prefixes are reserved by ODM (see below) and should not be used in mining object names.
- **ODM Reserved Prefixes:** The prefixes `DM$` and `DM_` are reserved for use by ODM across all schema object names in a given Oracle instance.

  Users must not directly access these ODM internal tables, that is, they should not execute any DDL, Query, or DML statements directly against objects named with these prefixes. Oracle recommends that you rename any existing objects in the database with these prefixes to avoid confusion in your application data management.
- **Input Data for Programs Using ODM:** All input data for ODM programs must be presented to ODM as an Oracle-recognized table, whether a view, table, or table function output.

2

ODM Java Programming

This chapter provides an overview of the steps required to perform basic Oracle Data Mining tasks and discusses the following topics related to writing data mining programs using the Java interface:

- The requirements for compiling and executing programs.
- How to perform common data mining tasks.

Detailed demo programs are provided as part of the installation.

2.1 Compiling and Executing ODM Programs

Oracle Data Mining depends on the following Java archive (`.jar`) files:

```
$ORACLE_HOME/dm/lib/odmapi.jar$ORACLE_HOME/jdbc/lib/ojdbc14.jar
$ORACLE_HOME/jlib/orai18n.jar
$ORACLE_HOME/lib/xmlparserv2.jar
```

These files must be in your `CLASSPATH` to compile and execute Oracle Data Mining programs.

2.2 Using ODM to Perform Mining Tasks

This section describes the steps required to perform several common data mining tasks using Oracle Data Mining. Data mining tasks are usually performed in a particular sequence. The following sequence is typical:

1. Collect and preprocess (bin or normalize) data. (This step is optional; ODM algorithms can automatically prepare input data.)
2. Build a model

3. Test the model and calculate lift (classification problems only)
4. Apply the model to new data

All work in Oracle Data Mining is done using `MiningTask` objects.

To implement a sequence of dependent task executions, you may periodically check the asynchronous task execution status using the `getCurrentStatus` method or block for completion using the `waitForCompletion` method. You can then perform the dependent task after completion of the previous task.

For example, follow these steps to perform the build, test, and compute lift sequence:

- Perform the build task as described in Section 2.2.2 below.
- After successful completion of the build task, start the test task by calling the execute method on a `ClassificationTestTask` or `RegressionTestTask` object. Either periodically check the status of the test operation or block until the task completes.
- After successful completion of the test task, execute the compute lift task by calling the execute method on a `MiningComputeLiftTask` object.

You now have (with a little luck) a model that you can use in your data mining application.

2.2.1 Prepare Input Data

Different algorithms require different preparation and preprocessing of the input data. Some algorithms require normalization; some require binning (discretization). In the Java interface the algorithms can prepare data automatically.

This section summarizes the steps required for different data preparation methodologies supported by the ODM Java API.

Automated Discretization (Binning) and Normalization

The ODM Java interface supports automated data preparation. If the user specifies active unprepared attributes, the data mining server automatically prepares the data for those attributes.

In the case of algorithms that use binning as the default data preparation, bin boundary tables are created and stored as part of the model. The model's bin boundary tables are used for the data preparation of the dataset used for testing or

scoring using that model. In the case of algorithms that use normalization as the default data preparation, the normalization details are stored as part of the model. The model uses those details for preparing the dataset used for testing or scoring using that model.

The algorithms that use binning as the default data preparation are Naive Bayes, Adaptive Bayes Network, Association, *k*-Means, and O-Cluster. The algorithms that use normalization are Support Vector Machines and Non-Negative Matrix Factorization. For normalization, the ODM Java interface supports only the automated method.

External Discretization (Binning)

For certain distributions, you may get better results if you bin the data before the model is built.

External binning consists of two steps:

- The user creates binning specification either explicitly or by looking at the data and using one of the predefined methods. For categorical attributes, there is only one method: Top-*N* Frequency. For numerical attributes, there are two methods: Equi-width and equi-width with winsorizing.
- The user bins the data following the specification created.

Specifically, the steps for external binning are as follows:

1. Create `DiscretizationSpecification` objects to specify the bin boundary specifications for the attributes.
2. Call `Transformation.createDiscretizationTables` method to create bin boundaries
3. Call `Transformation.discretize` method to discretize/bin the data.

Note that in the case of external binning, the user needs to bin the data consistently for all build, test, apply, and lift operations.

Embedded Discretization (Binning)

Embedded binning allows users to define their own customized automated binning. The binning strategy is specified by providing a bin boundary table that is produced by the bin specification creation step of external binning.

Specifically, the steps for embedded binning are as follows:

1. Create `DiscretizationSpecification` objects to specify the bin boundary specifications for the attributes.

2. Call the `Transformation.createDiscretizationTables` method to create bin boundaries.
3. Call the `setUserSuppliedDiscretizationTables` method in the `LogicalDataSpecification` object to attach the user created bin boundaries tables with the mining function settings object.

Keep in mind that because binning can have an effect on a model's accuracy, it is best when the binning is done by an expert familiar with the data being binned and the problem to be solved. However, if there is no additional information that can inform decisions about binning or if what is wanted is an initial exploration and understanding of the data and problem, ODM can bin the data using default settings, either by explicit user action or as part of the model build.

ODM groups the data into 5 bins by default. For categorical attributes, the 5 most frequent values are assigned to 5 different bins, and all remaining values are assigned to a 6th bin. For numerical attributes, the values are divided into 5 bins of equal size according to their order.

After the data is processed, you can build a model.

For an illustration of binning, see Appendix A.

2.2.2 Build a Model

This section summarizes the steps required to build a model.

1. Prepocess and prepare the input data as required.
2. Construct and store a `MiningFunctionSettings` object.
3. Construct and store a `MiningBuildTask` object.
4. Call the execute method; the execute method queues the work for asynchronous execution and returns an execution handle to the caller.
5. Periodically call the `getCurrentStatus` method to get the status of the task. Alternatively, use the `waitForCompletion` method to wait until all asynchronous activity for task completes.

After successful completion of the task, a model object is created in the database.

2.2.3 Find and Use the Most Important Attributes

Models based on data sets with a large number of attributes can have very long build times. To minimize build time, you can use ODM Attribute Importance to identify the critical attributes and then build a model using only these attributes.

Build an Attribute Importance Model

Identify the most important attributes by building an Attributes Importance model as follows:

1. Create a Physical Data Specification for input data set.
2. Discretize (bin) the data if required.
3. Create and store mining settings for the Attribute Importance.
4. Build the Attribute Importance model.
5. Access the model and retrieve the attributes by threshold.

Build a Model Using the Selected Attributes

After identifying the important attributes, build a model using the selected attributes as follows:

1. Access the model and retrieve the attributes by threshold or by rank.
2. Modify the Data Usage Specification by calling the function `adjustAttributeUsage` defined on `MiningFunctionSettings`. Only the attributes returned by Attribute Importance will be active for model building.
3. Build a model using the new Mining Function Settings.

2.2.4 Test the Model

This section summarizes the steps required to test a classification or a regression model.

1. Preprocess the test data as required. Test data must have all the active attributes used in the model and the target attribute in order to assess the model's accuracy.
2. Prepare (bin or normalize) the input data the same way the data was prepared for building the model.
3. Construct and store a task object. For classification problems, use `ClassificationTestTask`; for regression, use `RegressionTestTask`.
4. Call the `execute` method; the `execute` method queues the work for asynchronous execution and returns an execution handle to the caller.
5. Periodically, call the `getCurrentStatus` method to get the status of the task. As an alternative, use the `waitForCompletion` method to wait until all asychronous activity for the task completes.

6. After successful completion of the task, a test result object is created in the DMS. For classification problems, the results are represented using `ClassificaionTestResult` object; for regression problems, results are represented using `RegressionTestResult` object.

2.2.5 Compute Lift

This section summarizes the steps required to compute lift using a classification model.

1. Lift operation is typically done using the test data. Data preparation steps described in the section above also apply to the lift operation.
2. Construct and store a `MiningLiftTask` object.
3. Call the `execute` method; the `execute` method queues the work for asynchronous execution and returns an execution handle to the caller.
4. Periodically, call the `getCurrentStatus` method to get the status of the task. As an alternative, use the `waitForCompletion` method to wait until all asychronous activity for the task completes.
5. After successful completion of the task, a `MiningLiftResult` object is created in the DMS.

2.2.6 Apply the Model to New Data

You make predictions by applying a model to new data, that is, by *scoring* the data.

Any table that you score (apply a model to) must have the same format as the table used to build the model. If you build a model using a table that is in multi-record (transactional) format , any table that you apply that model to must be in multi-record format. Similarly, if the table used to build the model was in nontransactional (single-record) format, any table to which you apply the model must be in nontransactional format.

Note that you can score a single record, which must also be in the same format as the table used to build the model.

The steps required to apply a classification, clustering, or a regression model are as follows:

1. Preprocess the apply data as required. The apply data must have all the active attributes that were present in creating the model.

2. Prepare (bin or normalize) the input data the same way the data was prepared for building the model. If the data was prepared using the automated option at build time, then the apply data is also prepared using the automated option and other preparation details from building the model.
3. Construct and store a `MiningApplyTask` object. The `MiningApplyOutput` object is used to specify the format of the apply output table.
4. Call the `execute` method; the `execute` method queues the work for asynchronous execution and returns an execution handle to the caller.
5. Periodically, call the `getCurrentStatus` method to get the status of the task. As an alternative, use the `waitForCompletion` method to wait until all asynchronous activity for the task completes.
6. After successful completion of the task, a `MiningApplyResult` object is created in the DMS and the apply output table/view is created at the user-specified name and location.

3

ODM Java API Basic Usage

This chapter describes how to use the ODM Java interface to write data mining applications in Java. Our approach in this chapter is to use a simple example to describe the use of different features of the API.

For detailed descriptions of the class and method usage, refer to the Javadoc that is shipped with the product. See the administrator's guide for the location of the Javadoc.

3.1 Connecting to the Data Mining Server

To perform any mining operation in the database, first create an instance of `oracle.dmt.odm.DataMiningServer` class. This instance is used as a proxy to create connections to a data mining server (DMS), and to maintain the connection. The DMS is the server-side, in-database component that performs the actual data mining operations within ODM. The DMS also provides a metadata repository consisting of mining input objects and result objects, along with the namespaces within which these objects are stored and retrieved.

In this step, we illustrate creating a `DataMiningServer` object and then logging in to get the connection. Note that there is a `logout` method to release all the resources held by the connection

```
// Create an instance of the DMS server and get a connection.
// The database JDBC URL, user_name, and password for data mining
// user schema
DataMiningServer dms = new DataMiningServer(
   "DB_URL",// JDBC URL jdbc:oracle:thin:@Host name:Port:SID
   "user_name",
   "password");
//Login to get the DMS connection
oracle.dmt.odm.Connection m_dmsConn = dms.login();
```

3.2 Describing the Mining Data

In the ODM Java interface, `oracle.dmt.odm.data.LocationAccessData` (LAD) and `oracle.dmt.odm.PhysicalDataSpecification` (PDS) classes are used for describing the mining dataset (table/view in the user schema). To represent single-record format dataset, use an instance of `NonTransactionalDataSpecification` class, and to represent multi-record format dataset, use `TransactionalDataSpecification` class. Both classes are inherited from the common super class `PhysicalDataSpecification`. For more information about the data formats, refer to *ODM Concepts*.

In this step, we illustrate creating LAD and PDS objects for both types of formats.

3.2.1 Creating LocationAccessData

`LocationAccessData` (LAD) class encapsulates the dataset location details. The following code describes the creation of this object.

```
// Create a LocationAccessData by specifying the table/view name
// and the schema name
LocationAccessData lad =
          new LocationAccessData("input table name", "schema name");
```

3.2.2 Creating NonTransactionalDataSpecification

The `NonTransactionalDataSpecification` class contains the `LocationAccessData` object and specifies the data format as single-record case. The following code describes the creation of this object.

```
// Create the actual NonTransactionalDataSpecification
PhysicalDataSpecification pds =
          new NonTransactionalDataSpecification(lad);
```

3.2.3 Creating TransactionalDataSpecification

The `TransactionalDataSpecification` class contains a `LocationAccessData` object; it specifies the data format as multi-record case and it specifies the column roles.

This dataset must contain three types of columns: Sequence-Id/case-id column to represent each case, attribute name column, and attribute value column. This format is commonly used when the data has a large number of attributes. For more information, refer to *ODM Concepts*. The following code illustrates the creation of this object.

```
// Create the actual TransactionalDataSpecification for transactional data.
PhysicalDataSpecification pds =
          new TransactionalDataSpecification(
                    "CASE_ID", //column name for sequence id
                    "ATTRIBUTES", //column name for attribute name
                    "VALUES", //column name for value
                    lad //Location Access Data
          );
```

3.3 MiningFunctionSettings Object

The class `oracle.dmt.odm.settings.function.MiningFunctionSettings` (MFS) is the common super class for all types of mining function settings classes. It encapsulates the details of function and algorithm settings, logical data, and data usage specifications. For more detailed information about logical data and data usage specification, refer to Javadoc documentation for `oracle.dmt.odm.data.LogicalDataSpecification` and `oracle.dmt.odm.settings.function.DataUsageSpecification`.

An MFS object is a named object that can be stored in the DMS. If no algorithm is specified, the underlying DMS selects the default algorithm and its settings for that function. For example, Naive Bayes is the default algorithm for classification function. In this step, the ODM Java interface has the following function settings classes and a list of associated algorithm settings classes with each function.

```
oracle.dmt.odm.settings.function.ClassificationFunctionSettings
    oracle.dmt.odm.settings.algorithm.NaiveBayesSettings (Default)
    oracle.dmt.odm.settings.algorithm.AdaptiveBayesNetworkSettings
    oracle.dmt.odm.settings.algorithm.SVMClassificationSettings

oracle.dmt.odm.settings.function.RegressionFunctionSettings
    oracle.dmt.odm.settings.algorithm.SVMRegressionSettings (Default)

oracle.dmt.odm.settings.function.AssociationRulesFunctionSettings
    oracle.dmt.odm.settings.algorithm.AprioriAlgorithmSettings (Default)

oracle.dmt.odm.settings.function.ClusteringFunctionSettings
    oracle.dmt.odm.settings.algorithm.KMeansAlgorithmSettings (Default)
    oracle.dmt.odm.settings.algorithm.OClusterAlgorithmSettings (Default)

oracle.dmt.odm.settings.function.AttributeImportanceFunctionSettings
    oracle.dmt.odm.settings.algorithm.MinimumDescriptionLengthSettings
    (Defaults)
```

```
oracle.dmt.odm.settings.function.FeatureExtractionFunctionSettings
    oracle.dmt.odm.settings.algorithm.NMFAlgorithmSettings
```

In this step, we illustrate the creation of a `ClassificationFunctionSettings` object using Naive Bayes algorithm.

3.3.1 Creating Algorithm Settings

The class `oracle.dmt.odm.settings.algorithm.MiningAlgorithmSettings` *i*s the common superclass for all algorithm settings. It encapsulates all the settings that can be tuned by a data-mining expert based on the problem and the data. ODM provides default values for algorithm settings; refer to the Javadoc documentation for more information about each the algorithm settings. For example, Naive Bayes has two settings: `singleton_threshold` and `pairwise_threshold`. The default values for both of these settings is 0.01.

In this step we create a `NaiveBayesSettings` object that will be used by the next step to create the `ClassificationFunctionSettings` object.

```
// Create the Naive Bayes algorithm settings by setting both the pairwise
// and singleton thresholds to 0.01.
NaiveBayesSettings nbAlgo = new NaiveBayesSettings(0.02f,0.02f);
```

3.3.2 Creating Classification Function Settings

An MFS object can be created in two ways: by using the constructor or by using `create and adjust` utility methods. If you have the input dataset, it is recommended that you use the `create` utility method because it simplifies the creation of this complex object.

In this example, the utility method is used to create a `ClassificationFunctionSettings` object for a dataset, which has all unprepared categorical attributes and an ID column. Here we use automated binning; for more information about data preparation, see

```
// Create classification function settings
ClassificationFunctionSettings mfs =
          ClassificationFunctionSettings.create(
                   m_dmsConn,       //DMS Connection
                   nbAlgo,          //NB algorithm settings
                   pds,             //Build data specification
                   "target_attribute_name", //Target column
```

```
                    AttributeType.categorical, //Target attribute type
                    DataPreparationStatus.unprepared //Default preparation status
                    );

//Set ID attribute as an inactive attribute
mfs.adjustAttributeUsage(new String[]{"ID"},AttributeUsage.inactive);
```

3.3.3 Validate and Store Mining Function Settings

Because the `MiningFunctionSettings` object is a complex object, it is a good practice to validate the correctness of this object before persisting it. If you use utility methods to create MFS, then it will be a valid object.

The following code illustrates validation and persistence of the MFS object.

```
// Validate and store the ClassificationFunctionSettings object
try {
 mfs.validate();
 mfs.store(m_dmsConn, "Name_of_the_MFS");
} catch(ODMException invalidMFS) {
  System.out.println(invalidMFS.getMessage());
  throw invalidMFS;
}
```

3.4 MiningTask Object

The class `oracle.dmt.odm.task.MiningTask` is the common superclass for all the mining tasks. This class provides asynchronous execution of mining operations in the database using `DBMS_JOBS`. For each execution of the task an `oracle.dmt.odm.task.ExecutionHandle` object is created. The `ExecutionHandle` object provides the methods to retrieve the status of the execution and utility methods like `waitForCompletion`, `terminate`, and `getStatusHistory`. Refer to the Javadoc API documentation of these classes for more information.

The ODM Java interface has the following task classes:

- `oracle.dmt.odm.task.MiningBuildTask`
  This class is used for building a mining model
- `oracle.dmt.odm.task.ClassificationTestTask`
  This class is used for testing a classification model

- `oracle.dmt.odm.task.RegressionTestTask`
  This class is used for testing a regression model
- `oracle.dmt.odm.task.CrossValidateTask`
  This class is used for testing a Naive Bayes model using cross validation
- `oracle.dmt.odm.task.MiningLiftTask`
  This class is used for computing lift in case of classification models
- `oracle.dmt.odm.task.MiningApplyTask`
  This class is used for scoring new data using the mining model
- `oracle.dmt.odm.task.ModelImportTask`
  This class is used for importing a PMML mining model to ODM Java API native model
- `oracle.dmt.odm.task.ModelExportTask`
  This class is used for exporting a ODM Java API native model to PMML mining model

3.5 Build a Mining Model

To build a mining model, the `MiningBuildTask` object is used. It encapsulates the input and output details of the model build operation.

In this step, we illustrate creation, storing, and executing the `MiningBuildTask` object and task execution status retrieval by using `ExecutionHandle` object.

```
// Create a build task and store it.
MiningBuildTask buildTask =
          new MiningBuildTask(
          pds,
          "name_of_the_input_MFS",
          "name_of_the_model");

// Store the task
buildTask.store(m_dmsConn, "name_of_the_build_task");

// Execute the task
ExecutionHandle execHandle = buildTask.execute(m_dmsConn);

// Wait for the task execution to complete
MiningTaskStatus status = execHandle.waitForCompletion(dmsConnection);
```

After the build task completes successfully, the model is stored in the DMS with a name specified by the user.

3.6 MiningModel Object

The class `oracle.dmt.odm.model.MiningModel` is the common superclass for all the mining models. It is a wrapper class for the actual model stored in the DMS. Each model class provides methods for retrieving the details of the models. For example, `AssociationRulesModel` provides methods to retrieve the rules from the model using different filtering criteria. Refer to Javadoc API documentation for more details about the model classes.

In this step, we illustrate restoring the `NaiveBayesModel` object and retrieve the `ModelSignature` object. The `ModelSignature` object specifies the input attributes required to apply data using a specific model.

```
//Restore the naïve bayes model
NaiveBayesModel nbModel =
           (NaiveBayesModel)SupervisedModel.restore(
                   m_dmsConn,
                   "name_of_the_model");

//Get the model signature
ModelSignature nbModelSignature = nbModel.getSignature();
```

3.7 Testing a Model

After creating the classification model, you can test the model to assess its accuracy and compute a confusion matrix using the test dataset.

In this step, we illustrate how to test the classification model using the `ClassificationTestTask` object and how to retrieve the test results using the `ClassificationTestResult` object.

3.7.1 Describe the Test Dataset

To test the model, a compatible test dataset is required. For example, if the model is built using single-record dataset, then the test dataset must be single-record dataset. All the active attributes and target attribute columns must be present in the test dataset.

To test a model, the user needs to specify the test dataset details using the `PhysicalDataSpecification` class.

```
//Create PhysicalDataSpecification
 LocationAccessData lad = new LocationAccessData(
                             "test_dataset_name",
```

```
                                "schema_name" );
      PhysicalDataSpecification pds =
        new NonTransactionalDataSpecification( lad );
```

3.7.2 Test the Model

After creating the `PhysicalDataSpecification` object, create a `ClassificationTestTask` instance by specifying the input arguments required to perform the test operation. Before executing a task, it must be stored in the DMS. After invoking `execute` on the task, the task is submitted for asynchronous execution in the DMS. To wait for the completion of the task, use `waitForCompletion` method.

```
          //Create, store & execute Test Task
     ClassificationTestTask testTask = new ClassificationTestTask(
                 pds, //test data specification
                 "name_of_the_model_to_be_tested",
                 "name_of_the_test_results_object" );
     testTask.store(m_dmsConn, "name_of_the_test_task");
     taskTask.execute(m_dmsConn);

    //Wait for completion of the Test task
    MiningTaskStatus taskStatus =
                taskTask.waitForCompletion(m_dmsConn);
```

3.7.3 Get the Test Results

After successful completion of the test task, you can restore the results object persisted in the DMS using the `restore` method. The `ClassificationTestResult` object has `get` methods for accuracy and confusion matrix. The `toString` method can be used to display the test results.

```
//Restore the test results
ClassificationTestResult testResult =
          ClassificationTestResult.restore(m_dmsConn, "name of the test
results");

//Get accuracy
double accuracy = testResult.getAccuracy();

//Get confusion matrix
ConfusionMatrix confMatrix = testResult.getConfusionMatrix();

//Display results
System.out.println(testResult.toString());
```

3.8 Lift Computation

Lift is a measure of how much better prediction results are using a model than could be obtained by chance. You can compute lift after the model is built successfully. You can compute lift using the same test dataset. The test dataset must be compatible with the model as described in Section 2.2.4.

In this step, we illustrate how to compute lift by using `MiningLiftTask` object and how to retrieve the test results using `MiningLiftResult` object.

3.8.1 Specify Positive Target Value

To compute lift, a positive target value needs to be specified. This value depends on the dataset and the data mining problem. For example, for a marketing campaign response model, the positive target value could be "customer responds to the campaign". In the Java interface, `oracle.dmt.odm.Category` class is used to represent the target value.

```
Category positiveCategory = new Category(
  "Display name of the positive target value",
  "String representation of the target value",
  DataType.intType //Data type
);
```

3.8.2 Compute Lift

To compute lift, create a `MiningLiftTask` instance by specifying the input arguments that are required to perform the lift operation. The user needs to specify the number of quantiles to be used. A quantile is the specific value of a variable that divides the distribution into two parts: those values that are greater than the quantile value and those values that are less. Here the test dataset records are divided into the user-specified number of quantiles and lift is computed for each quantile.

```
     //Create, store & execute Lift Task
MiningLiftTask liftTask = new MiningLiftTask (
             pds, //test data specification
             10, //Number of quantiles
             positiveCategory, //Positive target value
             "name_of_the_input_model",
             "name_of_the_lift_results_object" );
liftTask.store(m_dmsConn, name_of_the_lift_task");
liftTask.execute(m_dmsConn);
```

```
//Wait for completion of the lift task
MiningTaskStatus taskStatus =
             liftTask.waitForCompletion(m_dmsConn);
```

3.8.3 Get the Lift Results

After successful completion of the test task, you can restore the results object persisted in the DMS using `restore method.MiningLiftResult`. To get the lift measures for each quantile use `getLiftResultElements()`. Method to`String()` can be used to display the lift results.

```
//Restore the lift results
MiningLiftResult liftResult =
          MiningLiftResult.restore(m_dmsConn, "name_of_the_lift_results");
//Get lift measures for each quantile
LiftResultElement[] quntileLiftResults =
          liftResult.getLiftResultElements()
//Display results
System.out.println(liftResult.toString());
```

3.9 Scoring Data Using a Model

A classification or clustering model can be applied to new data to make predictions; the process is referred to as "scoring data."

Similar to the test dataset, the apply dataset must have all the active attributes that were used to build the model. Unlike test dataset, apply dataset does not have a target attribute column; the apply process predicts the values of the target attribute. ODM Java API supports real-time scoring in addition to batch scoring (i.e., scoring with an input table)

In this step, we illustrate how to apply a model to a table/view to make predictions and how to apply a model to a single record for real-time scoring.

3.9.1 Describing Apply Input and Output Datasets

The Apply operation requires an input dataset that has all the active attributes that were used to build the model. It produces an output table in the user- specified format.

```
//Create PhysicalDataSpecification
LocationAccessData lad = new LocationAccessData(
                                  "apply_input_table/view_name",
                                  "schema_name"
```

```
                                );
PhysicalDataSpecification pds =
          new NonTransactionalDataSpecification( lad );

//Output table location details
LocationAccessData outputTable =  new LocationAccessData(
          "apply_output_table/view_name",
           "schema_name" );
```

3.9.2 Specify the Format of the Apply Output

The DMS also needs to know the content of the scoring output. This information is captured in a `MiningApplyOutput` (MAO) object. An instance of `MiningApplyOutput` specifies the data (columns) to be included in the apply output table that is created as the result of an apply operation. The columns in the apply output table are described by a combination of `ApplyContentItem` objects. These columns can be either from the input table or generated by the scoring task (for example, prediction and probability). The following steps create a `MiningApplyOutput` object:

```
// Create MiningApplyOutput object using default settings
MiningApplyOutput mao = MiningApplyOutput.createDefault();

// Add all the source attributes to be returned with the scored result.
// For example, here we add attribute "CUST_ID" from the original table
// to the apply output table
MiningAttribute sourceAttribute =
          new MiningAttribute("CUST_ID", DataType.intType,
                              AttributeType.notApplicable);
Attribute destinationAttribute = new Attribute(
               "CUST_ID",DataType.intType);

ApplySourceAttributeItem m_ApplySourceAttributeItem =
   new ApplySourceAttributeItem(sourceAttribute,destinationAttribute);
// Add a source and destination mapping
mao.addItem(m_ApplySourceAttributeItem);
```

3.9.3 Apply the Model

To apply the model, create a `MiningApplyTask` instance by specifying the input arguments that are required to perform the apply operation.

```
//Create, store & execute apply Task
MiningApplyTask applyTask = new MiningApplyTask(
```

```
                    pds, //test data specification
                    "name_of_the_model", //Input model name
                    mao, //MiningApplyOutput object
                    outputTable, //Apply output table location details
                    "name_of_the_apply_results" //Apply results name
                    );

applyTask.store(m_dmsConn, name_of_the_apply_task");
applyTask.execute(m_dmsConn);

//Wait for completion of the apply task
MiningTaskStatus taskStatus =
              applyTask.waitForCompletion(m_dmsConn);
```

3.9.4 Real-Time Scoring

To apply the model to a single record, use the `oracle.dmt.odm.result.RecordInstance` class. Model classes that support record apply have the static apply method, which can take `RecordInstance` object as input and returns with the prediction and probability.

In this step, we illustrate the creation of the `RecordInstance` object and score using Naive Bayes model's apply static method.

```
//Create RecordInstance object for a model with two active attributes
RecordInstance inputRecord = new RecordInstance();

//Add active attribute values to this record
AttributeInstance attr1 = new AttributeInstance("Attribute1_Name", value);
AttributeInstance attr2 = new AttributeInstance("Attribute2_Name", value);
inputRecord.addAttributeInstance(attr1);
inputRecord.addAttributeInstance(attr2);

//Record apply, output record will have the prediction value and its probability
value
RecordInstance outputRecord = NaiveBayesModel.apply(
m_dmsConn, inputRecord, "model_name");
```

3.10 Use of CostMatrix

The class `oracle.dmt.odm.CostMatrix` is used to represent the costs of the false positive and false negative predictions. It is used for classification problems to specify the costs associated with the false predictions. A user can specify the cost

matrix in the classification function settings. For more information about the cost matrix, see ODM Concepts.

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

```
// Define a list of categories
Category negativeCat = new Category("negativeResponse", "0",
                                    DataType.intType);
Category positiveCat = new Category("positiveResponse", "1",
                                    DataType.intType);

// Define a Cost Matrix
// AddEntry( Actual Category, Predicted Category, Cost Value)
CostMatrix costMatrix = new CostMatrix();
// Row 1
costMatrix.addEntry(negativeCat, negativeCat, new Integer("0"));
costMatrix.addEntry(negativeCat, positiveCat, new Integer("1"));
// Row 2
costMatrix.addEntry(positiveCat, negativeCat, new Integer("2"));
costMatrix.addEntry(positiveCat, positiveCat, new Integer("0"));
// Set Cost Matrix to MFS
mfs.setCostMatrix(costMatrix);
```

3.11 Use of PriorProbabilities

The class `oracle.dmt.odm.PriorProbabilities` is used to represent the prior probabilities of the target values. It is used for classification problems if the actual data has a different distribution for target values than the data provided for the model build. A user can specify the prior probabilities in the classification function settings. For more information about the prior probabilities, see ODM Concepts.

The following code illustrates how to create `PriorProbabilities` object, when the target has two classes: YES (1) and NO (0), and probability of YES is 0.05, probability of NO is 0.95.

```
// Define a list of categories
Category negativeCat = new Category(
        "negativeResponse", "0", DataType.intType);
Category positiveCat = new Category(
        "positiveResponse", "1", DataType.intType);
// Define a Prior Probability
```

```
// AddEntry( Target Category, Probability Value)
PriorProbabilities priorProbability = new PriorProbabilities();
// Row 1
priorProbability.addEntry(negativeCat, new Float("0.95"));
// Row 2
priorProbability.addEntry(positiveCat, new Float("0.05"));
// Set Prior Probabilities to MFS
  mfs.setPriors(priorProbability);
```

3.12 Data Preparation

Data Mining algorithms require the data to be prepared to build mining models and to score. Data preparation requirements can be specific to a function and an algorithm. ODM algorithms require binning (discretization) or normalization, depending on the algorithm. For more information about which algorithm requires what type of data preparation, see ODM Concepts. Java API supports automated binning, automated normalization, external binning, winsorizing, and embedded binning.

In this section, we illustrate how to do the automated binning, automated normalization, external binning, and embedded binning.

3.12.1 Automated Binning and Normalization

In the `MiningFunctionSettings`, if any of the active attributes are set as unprepared attributes, the DMS chooses the appropriate data preparation (i.e., binning or normalization), depending on the algorithm, and prepares the data automatically before sending it to the algorithm codes.

3.12.2 External Binning

The class `oracle.dmt.odm.transformation.Transformation` provides the utility methods to perform external binning. Binning is a two-step process, first bin boundary tables need to be created and then bin the actual data using the bin boundary tables as input.

The following code illustrates the creation of bin boundary tables for a table with one categorical attribute and one numerical attribute.

```
//Create an array of DiscretizationSpecification
//for the two columns in the table
DiscretizationSpecification[] binSpec = new DiscretizationSpecification[2];

//Specify binning criteria for categorical column.
```

```
//In this example we are specifying binning criteria
//as top 5 frequent values need to be used and
//the rest of the less frequent values need
//to be treated as OTHER_CATEGORY

CategoricalDiscretization binCategoricalCriteria =
        new CategoricalDiscretization(5,"OTHER_CATEGORY");

binSpec[0] = new DiscretizationSpecification(
        "categorical_attribute_name", binCategoricalCriteria);

//Specify binning criteria for numerical column.
//In this example we are specifying binning criteria
//as use equal width binning with 10 bins and use
//winsorize technique to filter 1 tail percent

float tailPercentage = 1.0f; //tail percentage value

NumericalDiscretization binNumericCriteria =
        new NumericalDiscretization(10, tailPercentage);
binSpec[1] = new DiscretizationSpecification(
        "numerical_attribute_name", binNumericCriteria);

 //Create PhysicalDataSpecification object for the input data
LocationAccessData lad = new LocationAccessData(
                                    "input_table_name",
                                    "schema_name" );
PhysicalDataSpecification pds =
        new NonTransactionalDataSpecification( lad );

//Create bin boundaries tables
Transformation.createDiscretizationTables(
        m_dmsConn, //DMS connection
        lad,  pds, //Input data details
        binSpec,   //Binning criteria
        "numeric_bin_boundaries_table",
        "categorical_bin_boundaries_table",
        "schema_name>");

//Resulting discretized view location
LocationAccessData resultViewLocation = new LocationAccessData(
          "output_discretized_view_name",
          "schema_name" );

//Perform binning
```

```
Transformation.discretize(
            m_dmsConn, // DMS connection
            lad, pds,  // Input data details
            "numeric_bin_boundaries_table",
            "categorical_bin_boundaries_table>,
            "schema_name",
            resultViewLocation, // location of the resulting binned view
            true               // open ended binning
            );
```

3.12.3 Embedded Binning

In case of external binning, the user needs to maintain the bin boundary tables and use these tables to bin the data. In case of embedded, the user can give the binning bin boundary tables as an input to the model build operation. The model will maintain these tables internally and use them for binning of the data for build, apply, test, or lift operations.

The following code illustrates how to associate the bin boundary tables with the mining function settings.

```
//Create location access data objects for bin boundary tables
LocationAccessData numBinBoundaries = new LocationAccessData(
            "numeric_bin_boundaries_table",
            "schema_name");

LocationAccessData catBinBoundaries = new LocationAccessData(
            "categorical_bin_boundaries_table>,
            "schema_name");

//Get the Logical Data Specification from the MiningFunctionSettings class
LogicalDataSpecification lds = mfs.getLogicalDataSpecification();

//Set the bin boundary tables to the logical data specification
lds.setUserSuppliedDiscretizationTables(numBinBoundaries, catBinBoundaries);
```

3.13 Text Mining

ODM Java API supports text mining for SVM and NMF algorithms. For these algorithms, an input table can have a combination of categorical, numerical, and text columns. The data mining server (DMS) internally performs the transformations required for the text data before building the model.

Note that for text mining, the case-id column must be specified in the NonTransactionalDataSpecification object, case-id column must have not-NULL unique values.

The following code illustrates how to set the text attribute in the ODM Java API.

```
//Set a caseid/sequenceid column for the dataset with active text attribute
Attribute sequenceAttr = new Attribute ("case_id_column_name", DataType.int);
pds.setSequenceAttribute( Attribute sequenceAttr )

//Set the text attribute
mfs.adjustAttributesType( new String[] {"text_attribute_column"},
            AttributeType.text );
```

3.14 Summary of Java Sample Programs

All the demo programs listed in the tables below are located in the directory $ORACLE_HOME/dm/demo/sample/java.

The summary description of these sample programs is also provided in $ORACLE_HOME/dm/demo/sample/java/README.txt.

Note: Before executing these programs, make sure that the SH schema and user schema are installed with the datasets used by these programs. You also need to provide DB URL, username, and password in login method and a valid data schema name by changing the DATA_SCHEMA_NAME constant value in the program.

Table 3–1 Java Sample Programs

Sample Program	Description
ABNDemo.java	Classification using the ABN algorithm
AIDemo.java	Determine most important attributes using the Attribute Importance algorithm; then use the resulting AI model to build classification model using Naive Bayes algorithm
ARDemo.java	Association (AR) model using the Apriori algorithm; extracting association rules
CostDemo.java	Use of cost matrix; compare results with and without the cost matrix
DataPrepDemo.java	Use of discretization methodologies: automated, external discretization, and user-supplied bin boundaries (embedded binning)
kMeansDemo.java	Clustering using the *k*-Means algorithm

Table 3–1 (Cont.) Java Sample Programs

Sample Program	Description
`NBDemo.java`	Classification using the Naive Bayes algorithm
`NMFDemo.java`	Feature extraction and text mining using the Non-Negative Matrix Factorization (NMF) algorithm
`OClusterDemo.java`	Clustering using the O-Cluster algorithm.
`PMMLDemo.java`	Import and export a PMML model
`PriorsDemo.java`	Use of prior probability; compare results with and without the prior probability
`SVMCDemo.java`	Classification and text mining using the SVM algorithm.
`SVMRDemo.java`	Regression using the SVM algorithm

4

DBMS_DATA_MINING

This chapter discusses the following topics related to writing data mining programs with the PL/SQL interface:

- The requirements for compiling and executing Oracle Data Mining programs.
- How to perform common data mining tasks using Oracle Data Mining.
- Tips and techniques for using the algorithms.

This chapter provides an overview of the steps required to perform basic Oracle Data Mining tasks. For detailed examples of how to perform these tasks, see the sample programs in Chapter 5.

This chapter does not include detailed descriptions of the PL/SQL subprograms. For that information, see the `DBMS_DATA_MINING` and `DBMS_DATA_MINING_TRANSFORM` chapters in th*e PL/SQL Packages and Types Reference*.

The `DBMS_DATA_MINING` package provides PL/SQL support for in-database data mining. You can use the package to build a mining model, test the model, and apply this model to your data to obtain predictive and descriptive information.

See also:

- *Oracle Data Mining Concepts.*
- `DBMS_DATA_MINING_TRANSFORM`, a supplied package that supports data preprocessing for mining (described in*PL/SQL Packages and Types Reference*).

This chapter discusses the following topics:

- Section 4.1, "Development Methodology"
- Section 4.2, "Mining Models, Function, and Algorithm Settings"
- Section 4.3, "Mining Operations and Results"

- Section 4.4, "Mining Data"
- Section 4.5, "Performance Considerations"
- Section 4.6, "Rules and Limitations for DBMS_DATA_MINING"
- Section 4.9, "Model Export and Import"

DBMS_DATA_MINING subprograms are presented in *PL/SQL Packages and Types Reference*. Sample code is described in Chapter 5 of this manual; the code itself is in the dm/demo/sample/plsql directory.

4.1 Development Methodology

The development methodology for data mining using the DBMS_DATA_MINING API is divided into two phases.

The first phase includes your application and data analysis and design, where you perform the following two steps:

1. Analyze your problem, and choose the mining function and algorithm.
2. Analyze the data to be used for building mining models (build data), testing predictive models (test data), and the new data on which the model will be applied (scoring data).

The second phase involves developing a mining application using DBMS_DATA_MINING and DBMS_DATA_MINING_TRANSFORM packages.

3. Prepare the build, test, and scoring data using the DBMS_DATA_MINING_TRANSFORM package or other third-party tool or direct SQL or PL/SQL utility scripts in a manner suitable for the chosen mining function and algorithm. An important caveat is that the three detests referred to above have to be prepared in an identical manner for mining results to be meaningful. This is an optional step.
4. Prepare a settings table that overrides the default mining algorithm for a given mining function, and the default algorithm settings. This is also an optional step.
5. Build a mining model for the given training dataset.
6. For predictive models (classification and regression), test the model for its accuracy and other attributes. This amounts to applying the model on the test data (i.e., scoring the test data), and computing various matrix on the apply results.

7. Retrieve the model signature to determine the mining attributes required by a given model for scoring. This information will help ascertain that the scoring data is suitable for a given model. This is an optional step.
8. Apply a classification, regression, clustering, or feature extraction model to new data to generate predictions and/or descriptive summaries and patterns about the data.
9. Retrieve the model details to understand why a model scored the data in a particular manner. This is an optional step.
10. Repeat steps 3 through 9 until you obtain satisfactory results.

4.2 Mining Models, Function, and Algorithm Settings

The DBMS_DATA_MINING package creates a mining model for a mining function using a specified mining algorithm that supports the function. The algorithm can be influenced by specific algorithm settings.

4.2.1 Mining Model

A model is identified by its name. Like tables in the database, a model has storage associated with it. But unlike a table, the form, shape, and content of this storage is opaque to the user. However, the user can view the contents of a model — that is, the patterns and rules that constitute a mining model — using algorithm-specific GET_MODEL_DETAILS functions. In addition, dm_user_models provides the model size in megabytes.

4.2.2 Mining Function

The DBMS_DATA_MINING package supports Classification, Regression, Association Rules, Clustering, and Feature Extraction. You can specify your choice of mining function through a parameter to the CREATE_MODEL procedure.

4.2.3 Mining Algorithm

Each mining function can be implemented using one or more algorithms. Table 4–1 provides a list of supported algorithms. Oracle assumes a default algorithm for each

mining function, but you can override this default through an explicit setting in the settings table.

Table 4–1 DBMS_DATA_MINING Summary of Functions and Algorithms

Mining Function	Mining Algorithm
Classification	Naive Bayes (NB) — default algorithm
Classification	Adaptive Bayes Network (ABN)
Classification	Support Vector Machine (SVM)
Regression	Support Vector Machine (SVM) — default algorithm
Association Rules	Apriori Association Rules (AR)
Clustering	*k*-Means (KM)
Feature Extraction	Non-Negative Matrix Factorization (NMF)

Each algorithm has one or more settings or parameters that influence the way it builds the model. Oracle assumes a default set of algorithm settings for each mining algorithm. These defaults are available for your review through the table function `GET_DEFAULT_SETTINGS`. To override the defaults, you must provide the choice of the algorithm and the settings for the algorithm through a settings table input to the `CREATE_MODEL` procedure.

4.2.4 Settings Table

The settings table is a simple relational table with a *fixed* schema. You can choose the name of the settings table, but the column names and their types must be defined as specified below.

```
(setting_name   VARCHAR2(30),
setting_value   VARCHAR2(128))
```

The values provided in the settings table override the default values assumed by the system. The values inserted into the *`setting_name`* column are one or more of several constants defined in the `DBMS_DATA_MINING` package. Depending on what the setting name denotes, the value for the *`setting_value`* column can be a predefined constant or the actual numerical value corresponding to the setting itself. The *`setting_valu`*e column is defined to be `VARCHAR2`, so you must cast numerical inputs to string using the `TO_CHAR()` function before input into the settings table.

Table 4–2 through Table 4–7 list the various setting names and the valid setting values, with a brief explanation of each setting

Table 4–2 DBMS_DATA_MINING Function Settings

Algorithm Settings	Setting Value (with Permissible Value Ranges)
`algo_name`	Classification: One of: ▪ `algo_naive_bayes` ▪ `algo_support_vector_machines` ▪ `algo_adaptive_bayes_network` Regression: ▪ `algo_support_vector_machines` Association Rules: ▪ `algo_apriori_association_rules` Clustering: ▪ `algo_kmeans` Feature Extraction: ▪ `algo_non_negative_matrix_factor` Attribute Importance: ▪ `algo_ai_mdl`
`clas_priors_table_name`	`VARCHAR2` string denoting the name of a relational table of fixed schema containing prior probabilities. The schema of this table is provided in Section 4.2.4.1 on page 4-10. This input is applicable only for classification algorithms. The prior probabilities table must be present in the current user's schema
`clus_num_clusters`	`TO_CHAR(`*`numeric_expr`*`>= 1)` Number of clusters generated by a clustering algorithm Default value is 10
`feat_num_features`	`TO_CHAR(`*`numeric_expr`*` >= 1)` Number of features to be extracted Default value estimated from the data by the algorithm
`asso_max_rule_length`	`TO_CHAR(2 <=` *`numeric_expr`* `<= 20)` Maximum rule length for AR algorithm Default value is 4

Table 4–2 DBMS_DATA_MINING Function Settings

Algorithm Settings	Setting Value (with Permissible Value Ranges)
asso_min_confidence	TO_CHAR(0 <= *numeric_expr* <= 1) Minimum confidence value for AR algorithm Default value is 0.1
asso_min_support	TO_CHAR(0 <= *numeric_expr* <= 1) Minimum support value for AR algorithm Default value is 0.1

Table 4–3 Algorithm Settings for Adaptive Bayes Network

Setting Name	Setting Value (with Permissible Value Ranges)
abns_model_type	Model type for Adaptive Bayes Network: ■ abns_single_feature (single feature) ■ abns_multi_feature (multi feature ■ abns_naive_bayes (naive bayes) Default value is abns_multi_feature
abns_max_build_minutes	TO_CHAR(*numeric_expr* >= 0) The maximum time threshold for completion of model build. Default value is 0, which implies no time limit.
abns_max_nb_predictors	TO_CHAR(*numeric_expr* > 0) Maximum number of Naive Bayes predictors to be considered for model build, when the model type is chosen to be abns_naive_bayes. Default value is 10.
abns_max_predictors	TO_CHAR(*numeric_expr* > 0) Default is 25

Table 4–4 Algorithm Settings for Naive Bayes

Setting Name	Setting Value (with Permissible Value Ranges)
nabs_singleton_threshold	TO_CHAR(0 <= *numeric_expr* <=1) Value of singleton threshold for NB algorithm Default value is 0.01
nabs_pairwise_threshold	TO_CHAR (0 <= *numeric_expr* <=1) Value of pairwise threshold for NB algorithm Default value is 0.01

Table 4–5 Algorithm Settings for Support Vector Machines

Setting Name	Setting Value (with Permissible Value Ranges)
svms_kernel_function	Kernel for Support Vector Machine: ▪ svms_linear (for Linear Kernel ▪ svms_gaussian (for Gaussian Kernel) Default value is svms_linear
svms_kernel_cache_size	TO_CHAR(*numeric_expr* > 0) Value of kernel cache size for SVM algorithm. Applies to Gaussian kernel only. Default value is 50000000 bytes
svms_conv_tolerance	TO_CHAR(*numeric_expr* > 0) Convergence tolerance for SVM algorithm Default value is 0.001.
svms_std_dev	TO_CHAR(*numeric_expr* > 0) Value of standard deviation for SVM algorithm This is applicable only for Gaussian kernel Default value estimated from the data by the algorithm
svms_complexity_factor	TO_CHAR(*numeric_expr* > 0) Value of complexity factor for SVM algorithm Default value estimated from the data by the algorithm

Table 4–5 (Cont.) Algorithm Settings for Support Vector Machines

Setting Name	Setting Value (with Permissible Value Ranges)
svms_epsilon	TO_CHAR(*numeric_expr* > 0) Value of epsilon factor for SVM Regression Default value estimated from the data by the algorithm

Table 4–6 Algorithm Settings for Non-Negative Matrix Factorization

Setting Name	Setting Value (with Permissible Value Ranges)
nmfs_random_seed	TO_CHAR*(numeric_expr)* Number of iterations for NMF algorithm Default value is –1
nmfs_num_iterations	TO_CHAR(1 <= *numeric_expr* <= 500) Number of iterations for NMF algorithm Default value is 50
nmfs_conv_tolerance	TO_CHAR(0 < *numeric_expr* <= 0.5) Convergence tolerance for NMF algorithm Default value is 0.05

Table 4–7 Algorithm Settings for k-Means

Setting Name	Setting Value (with Permissible Value Ranges)
kmns_distance	Distance Function for *k*-Means Clustering: ■ kmns_euclidean ■ kmns_cosine ■ kmns_fast_cosine Default value is kmns_euclidean
kmns_iterations	TO_CHAR(0 < *numeric_expr* <= 20) Number of iterations for *k*-Means algorithm Default value is 3
kmns_conv_tolerance	TO_CHAR(0 < *numeric_expr* <= 0.5) Convergence tolerance for *k*-Means algorithm Default value is 0.01

Table 4–7 (Cont.) Algorithm Settings for k-Means

Setting Name	Setting Value (with Permissible Value Ranges)
kmns_split_criterion	Split criterion for *k*-Means Clustering: ■ kmns_variance ■ kmns_size Default value is kmns_variance
kmns_num_bins	Number of histogram bins. Specifies the 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. Range > 0 Default value is 10.
kmns_block_growth	TO_CHAR(1 < *numeric_expr* <= 5) Growth factor for memory allocated to hold cluster data Default value is 2
kmns_min_pct_attr_support	Minimum percentage support required for attributes in rules. Specifies the minimum percentage of values for an attribute in a given cluster required to include this attribute in the rule description of the cluster. That is, if the required support level is not met, the attribute would be omitted from the rule. This would allow retaining in the rule only the well-represented attributes. Setting the parameter value too high in data with missing values can result in very short or even empty rules. Range >= 0 and <= 1 Default is 0.1.

You can create a settings table as shown in the example below for an SVM classification model, and edit the individual values using SQL DML.

```
CREATE TABLE drugstore_settings (
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(128))

BEGIN
```

```
-- override the default for convergence tolerance for SVM Classification
INSERT INTO drugstore_model_settings (setting_name, setting_value)
VALUES (dbms_data_mining.svms_conv_tolerance, TO_CHAR(0.081));
COMMIT;
END;
```

The table function GET_DEFAULT_SETTINGS provides you all the default settings for mining functions and algorithms. If you intend to override all the default settings, you can create a seed settings table and edit them using DML.

```
BEGIN
CREATE TABLE drug_store_settings AS
SELECT setting_name, setting_value
  FROM TABLE (DBMS_DATA_MINING.GET_DEFAULT_SETTINGS
 WHERE setting_name LIKE 'SVMS_%';
-- update the values using appropriate DML
END;
```

You can also create a settings table based on another model's settings using GET_MODEL_SETTINGS, as shown in the example below.

```
BEGIN
CREATE TABLE my_new_model_settings AS
SELECT setting_name, setting_value
  FROM TABLE (DBMS_DATA_MINING.GET_MODEL_SETTINGS('my_other_
model'));
END;
```

4.2.4.1 Prior Probabilities Table

Consult *Oracle Data Mining Concepts* for an explanation of the prior probabilities table. You can specify a prior probabilities table as an optional function setting when building classification models.

You must create the prior probabilities table using the fixed schema shown below. For numerical targets, use the following schema:

```
target_value      NUMBER
prior_probability NUMBER
```

For categorical targets, use the following schema:

```
target_value      VARCHAR2
prior_probability NUMBER
```

Next, provide the name of the prior probabilities table as input to the *setting_value* column in the settings table, with the corresponding value for the *setting_name* column to be DBMS_DATA_MINING.clas_priors_table_name, as shown below:

```
BEGIN
INSERT INTO drugstore_settings (setting_name, setting_value) VALUES (DBMS_
DATA_MINING.class_priors_table_name,
'census_priors');
COMMIT;
END;
```

4.2.4.2 Cost Matrix Table

Consult *Oracle Data Mining Concepts* for an explanation of the cost matrix. You must create a cost matrix table with the fixed schema shown below. For numerical targets, use the following schema:

```
actual_target_value    NUMBER
predicted_target_value NUMBER
cost                   NUMBER
```

For categorical targets, use the following schema:

```
actual_target_value    VARCHAR2
predicted_target_value VARCHAR2
cost                   NUMBER
```

The DBMS_DATA_MINING package enables you to evaluate the cost of predictions from classification models in an iterative manner during the experimental phase of mining, and to eventually apply the optimal cost matrix to predictions on the actual scoring data in a production environment.

The data input to each COMPUTE procedure in the package is the result generated from applying the model on test data. In addition, if you also provide a cost matrix as an input, the COMPUTE procedure generates test results taking the cost matrix into account. This enables you to experiment with various costs for a given prediction against the same APPLY results, without rebuilding the model and applying it against the same test data for every iteration.

Once you arrive at an optimal cost matrix, you can then input this cost matrix to the RANK_APPLY procedure along with the results of APPLY on your scoring data. RANK_APPLY will provide your new data ranked by cost.

4.3 Mining Operations and Results

There are essentially four classes of mining operations supported by the DBMS_DATA_MINING package:

- Operations that create, drop, and rename a model: CREATE_MODEL, DROP_MODEL, RENAME_MODEL.
- Operation that reads a model to score new data: APPLY.
- Operation that enables ranking of APPLY results or other data that is presented using the same schema as that of APPLY results: RANK_APPLY.
- Operations that read and describe a model: GET_MODEL_DETAILS, GET_MODEL_SETTINGS, GET_MODEL_SIGNATURE.
- Operations that help test a classification model, based on the results of an APPLY operation on the test data, or based on any other data that is presented using the same schema as that of the APPLY results: COMPUTE_CONFUSION_MATRIX, COMPUTE_LIFT, and COMPUTE_ROC.
- Operations that help transport a model from one schema to another, or from one database instance to another: EXPORT_MODEL, IMPORT_MODEL.
- GET_DEFAULT_SETTINGS returns default values for all the settings.

Of these, the first set represents DDL-like operations. The last set represents utilities. The rest are query-like operations in that they do not modify the model. Besides these operations, the following capabilities are also provided as part of the Oracle Data Mining installation:

- User Views — DM_USER_MODELS
- Queries to compute metrics that test regression models.

Mining results are either returned as result sets or persisted as fixed schema tables.

4.3.1 Build Results

The CREATE_MODEL operation creates a mining model. The viewable contents of a mining model are provided to you through GET_MODEL_DETAILS functions for each supported algorithm. In addition, GET_MODEL_SIGNATURE and GET_MODEL_SETTINGS provide descriptive information about the model.

4.3.2 Apply Results

The APPLY operation creates and populates a fixed schema table with a given name. The schema of this table varies based on the particular mining function, algorithm, and target attribute type — numerical or categorical.

The RANK_APPLY operation takes this results table as input and generates another table with results ranked based on a top-N input, and for classification models, also based on cost. The schema of this table varies based on the particular mining function, algorithm, and the target attribute type — numerical or categorical.

4.3.3 Test Results for Classification Models

The COMPUTE routines provided in the package are the most popularly used metrics for classification. They are not tied to a particular model — they can compute the metrics from any meaningful data input as long as the schema of the input tables fits the specification of the apply results table and the targets tables. Please consult any of the COMPUTE descriptions in this document for details.

4.3.4 Test Results for Regression Models

The most commonly used metrics for regression models are root mean square error and mean absolute error. You can use the simple SQL queries provided below to compute these metrics by replacing the italicized tokens with table and column names appropriate for your application.

4.3.4.1 Root Mean Square Error

```
SELECT sqrt(avg((A.prediction - B.target_column_name) *
                (A.prediction - B.target_column_name))) rmse
  FROM apply_results_table A, targets_table B
 WHERE A.case_id_column_name = B.case_id_column_name;
```

4.3.4.2 Mean Absolute Error

Given the targets_table generated from the test data with the schema:

```
(case_id_column_name    VARCHAR2,
target_column_name      NUMBER)
```

and apply results table for regression with the schema:

```
(case_id_column_name    VARCHAR2,
prediction              NUMBER)
```

and a normalization table (optional) with the schema:

```
(attribute_name        VARCHAR2(30),
scale                  NUMBER,
shift                  NUMBER)
```

the query for mean absolute error is:

```
SELECT  /*+PARALLEL(T) PARALLEL(A)*/
    AVG(ABS(T.actual_value - T.target_value)) mean_absolute_error
  FROM (SELECT B.case_id_column_name
               (B.target_column_name * N.scale + N.shift) actual_value
          FROM targets_table B,
               normalization_table N
         WHERE N.attribute_name = B.target_column_name AND
               B.target_column_name = 1) T,
       apply_results_table_name A
 WHERE A.case_id_column_name = T.case_id_column_name;
```

You can fill in the italicized values with the actual column and table names chosen by you. If the data has not undergone normalization transformation, you can eliminate those references from the subquery. See `dm/demo/sample/plsql/svmrdemo.sql` for an example.

4.4 Mining Data

The data input for all the mining operations should be through standard relational tables, where each row in the table represents a case, and each column in the table uniquely represents a mining attribute. We call this format *single-record case*.

A table in the Oracle RDBMS supports 1000 columns. The `DBMS_DATA_MINING` package requires a mandatory case identifier column, which implies that you can provide 999 mining attributes using this representation, which is adequate for most business applications.

4.4.1 Wide Data Support

The notable exceptions to the common scenario are applications in the domains of bioinformatics, text mining, and such specialized areas where data is characterized to be *wide* and *shallow* — with relatively few cases numbering in the thousands, but with several thousand mining attributes.

You can provide such data in a *multi-record case* format, where *`attribute`*, *`value`* pairs are grouped into collections (nested tables) associated with a given case-id.

You must use the fixed collection types DM_Nested_Numericals and DM_Nested_Categoricals to define columns that represent collections of numerical attributes and categorical attributes respectively.

A few caveats on the use of multi-record case format:

- You can intersperse columns defined to be of the two types referred to above with scalar columns that represent individual attributes in a table or view.
- For a given case-id, attribute names have to be unique across all the collections and individual columns.
- Target (for classification, regression) must be a non-nested attribute. We cannot accept a target attribute represented in a nested column or the nested column itself.

From a data modeling perspective, the multi-record case format is analogous to the dimension-fact relationship commonly found in OLAP applications, where the dimension and fact tables are aggregated based on a common key for generating reports using materialized views or SQL queries.

The main distinction between the OLAP fact table and the multi-record case is this: Each row in the fact table represents a column value, whereas each row in the multi-record collection represents an attribute name (paired with its corresponding value in another column in the nested table). The two fixed collection types are provided to highlight and enforce this distinction. These two types are defined with the reasonable assumption that mining attributes of the same type (numericals versus categoricals) are generally grouped together, just as a fact table contains values that logically correspond to the same entity.

Oracle strongly recommends that you present your multi-record case data using object views, and present the view as input to CREATE_MODEL and APPLY operations. Apart from the benefit of providing all your mining attributes through a single row-source without impacting their physical data storage, the view acts as a join specification on the underlying tables that can be used by the server for efficiently accessing your data.

We illustrate this discussion on wide data with a real-world example of an analytical pipeline for brain tumor research. The datasets of relevance are gene expression data from the instruments (fact table), and the clinical data about the patient (dimension table). The schemas of these tables are provided below.

4.4.1.1 Clinical Data — Dimension Table

```
(case_id    NUMBER,
name        VARCHAR2(30)
type        VARCHAR2(30)
subtype     VARCHAR2(30),
gender      CHAR(1),
age         NUMBER,
status      VARCHAR2(30))
```

4.4.1.2 Gene Expression Data — Fact Table

```
(case_id    NUMBER,
gene        VARCHAR2(30),
expr        NUMBER)
```

Let us consider building a model with `status` as the target prediction, and with `sex`, `age`, and `expr` being the predictors. You can provide the build data input using an object view that combines the `clinical_data` table and the `gene_expression_data` table with the schema:

```
(case_id     NUMBER,
gender       CHAR(1),
age          NUMBER,
gene_expr    DM_Nested_Numericals,
status       VARCHAR2(30))
```

The query for constructing such an object view is relatively simple in Oracle SQL:

```
CREATE OR REPLACE VIEW gene_expr_build AS
SELECT C.case_id,
       C.gender,
       C.age,
       CAST(MULTISET(
       SELECT gene, expr
         FROM gene_expression_data
        WHERE case_id = C.case_id) AS DM_Nested_Numericals
       ) gene_expr,
       C.status
 FROM  clinical_data C
```

Now `gene_expr_build` can be provided as the input *`data_table_name`* for `CREATE_MODEL`.

4.4.2 Attribute Types

Oracle Data Mining handles *categorical* and *numerica*l attributes. See *Oracle Data Mining Concepts* for explanation and examples of attribute types.

The DBMS_DATA_MINING package infers the attribute type based on the type of the column representing the mining attribute. You must define all columns representing numerical attributes to be of type NUMBER. You must define all columns representing categorical attributes to be of type VARCHAR2 or CHAR.

In situations where you have numbers that are deemed categorical data by your application logic, you must typecast such attribute values using the TO_CHAR() operator and populate them into a VARCHAR2 or CHAR column representing the mining attribute.

In situations where you have numeric attribute values stored in a CHAR or VARCHAR2 column, you must typecast those attribute values using the TO_NUMBER() operator and store them in a NUMBER column.

If persisting these transformed values in another table is not a viable option, you could also create a view with these conversions in place, and provide the view name to represent the training data input for the CREATE_MODEL operation.

4.4.3 Target Attribute

Classification and Regression algorithms require a target attribute. The package supports single targets. The target attribute for all classification algorithms can be numerical or categorical. SVM Regression supports only numerical target attributes.

4.4.4 Data Transformations

All mining operations assume the incoming data to be already prepared and transformed. You can preprocess your data using the DBMS_DATA_MINING_TRANSFORM package, or by using any third-party tool, or using your own homegrown utilities based on SQL and/or PL/SQL scripts.

If you preprocess or transform the input data, you must also reverse-transform the results from APPLY to map the data back to its original form.

See *PL/SQL Packages and Types Reference* for a description of the DBMS_DATA_MINING_TRANSFORM package.

The suggestion to customers with wide data is to perform transforms before forming WD views on data in single-record (2D) or multi-record (transactional) format. It is possible to use dbmsdmxf for multi-record format. In this case, all

attributes are transformed in a similar way. In most of the cases, attributes in transactional (multi-record) form are of the same scale and thus this approach works. Otherwise, our suggestion is to split the data into sets of similar items and then transform them separately.

4.5 Performance Considerations

If you have mining attributes numbering in the few hundreds, and your application requires them to be represented as columns in the same row of the table, here are some data storage issues to consider. For a table with several columns, the key question to consider is the (average) row length, not the number of columns. Having more than 255 columns in a table built with a smaller block size typically results in intra-block chaining. Oracle stores multiple row pieces in the same block owing to pragmatics of design, but the overhead to maintain the column information is minimal as long as all row pieces fit in a single data block. If the rows don't fit in a single data block, you may consider using a larger database block size (or use multiple block sizes in the same database). For more details, consult the *Oracle Data Mining Concepts* and the *Oracle Database Performance Tuning Guide*.

4.6 Rules and Limitations for DBMS_DATA_MINING

The use of `DBMS_DATA_MINING` is subject to the following rules and limitations:

- The `CREATE_MODEL` and `APPLY` operations require a case identifier column of type `VARCHAR2`, `CHAR`, or `NUMBER`. The length of the case identifier values is limited to 128 bytes.
- The model name must not be greater than 25 bytes in length, must not be enclosed in quotes, and should not contain any special characters except underscores ('_'). In other words, `DBMS_DATA_MINING` does not treat model names in a case-sensitive manner like Oracle schema objects and up-cases all model names internally.
- All schema objects with prefix `DM_`, `DM$P`, `DM$J`, `DM$T` are system objects that should not be read or updated by the user. Direct queries against these tables may be possible, but the queries may provide meaningless results. Direct DML against these tables can irrevocably corrupt your model. Hence Oracle strongly recommends that you do not perform any operations on these system objects.
- Oracle Data Mining does not support a general privilege model that spans multiple users. `GRANT` and `REVOKE` of read and update privileges on a mining model across user schemas are not yet supported. The user can only read and update models that are generated in a given schema owned by that user.

Models in one schema or database instance can be exported using EXPORT_MODEL to other schemas or database instances.

- As a corollary, results of all mining operations are restricted to being generated in the schema corresponding to the user session from which a particular mining operation is launched.
- In any given session, you must commit all DML to schema objects before invoking operations in the DBMS_DATA_MINING package.

4.7 Summary of Data Types, Constants, Exceptions, and User Views

The DBMS_DATA_MINING and the DBMS_DATA_MINING_TRANSFORM packages use the data types shown in Table 4–8.

Table 4–8 DBMS_DATA_MINING Summary of Data Types

Data Type	Purpose
DM_ABN_Detail	This type represents each row of the model detail output generated by GET_MODEL_DETAILS_ABN
DM_ABN_Details	This type represents the ABN model details generated by GET_MODEL_DETAILS_ABN
DM_Centroid	This type represents the centroid of a cluster. It is used when retrieving cluster details using GET_MODEL_DETAILS_KM.
DM_Child	This type represents each child node of a cluster
DM_Children	This type represents a set of children nodes for a given cluster identifier
DM_Cluster	This type represents a cluster retrieved using GET_MODEL_DETAILS_KM
DM_Clusters	This type represents a set of clusters
DM_Conditional	This type represents each conditional probability from a set of conditional probabilities associated with each mining attribute used in a Naive Bayes or Adaptive Bayes Network model
DM_Conditionals	This type represents conditional probabilities associated with a given mining attribute used in a Naive Bayes or Adaptive Bayes Network model. It is used when retrieving model details using GET_MODEL_DETAILS_NB or GET_MODEL_DETAILS_ABN respectively.

Table 4–8 DBMS_DATA_MINING Summary of Data Types

Data Type	Purpose
DM_Histogram_Bin	This type represents a histogram associated with a cluster identifier. It is used when retrieving cluster details using GET_MODEL_DETAILS_KM
DM_Histograms	This type represents a set of histograms
DM_Item	This type represents an item in a set of items
DM_Items	This type represents the set of items in an ItemSet
DM_ItemSet	This type represents an ItemSet
DM_ItemSets	This type represents frequent Itemsets in Association models.
DM_Model_Settings	This type represents the algorithm settings retrieved using the GET_MODEL_SETTINGS function.
DM_Model_Signature	This type represents a list of model signature attributes generated by GET_MODEL_SIGNATURE
DM_Modelname_List	This type represents a list of model names provided as input for the parameter model_names in EXPORT_MODEL and IMPORT_MODEL procedures.
DM_NB_Detail	his type represents the each row of the model detail output generated by GET_MODEL_DETAILS_NB
DM_NB_Details	This type represents the NB model details generated by GET_MODEL_DETAILS_NB.
DM_Nested_Categoricals	This type represents a nested table of categorical attributes, used for representing wide data.
DM_Nested_Numericals	This type represents a nested table of numerical attributes, used for representing wide data.
DM_NMF_Attribute	This type represents each attribute in an attribute set for NMF model details
DM_NMF_Attribute_Set	This type represents a set of attributes that correspond to a feature identifier, returned by GET_MODEL_DETAILS_NMF.
DM_NMF_Feature	This type represents a feature in a NMF model
DM_NMF_Feature_Set	This type represents a set of features returned by GET_MODEL_DETAILS_NMF.
DM_Predicate	This type represents each predicate in the set of predicates in a rule.

Table 4–8 DBMS_DATA_MINING Summary of Data Types

Data Type	Purpose
DM_Predicates	This type represents a set of predicates that constitute either the antecedent or the consequent of a rule.
DM_Ranked_Attribute	This type represents an entry in the set of ranked attribute returned by GET_MODEL_DETAILS_AI, ranked by the attribute's importance.
DM_Ranked_Attributes	This type represents a list of ranked attributes returned by GET_MODEL_DETAILS_AI.
DM_Rule	This type represents each rule in a list of rules generated by either GET_ASSOCIATION_RULES or GET_MODEL_DETAILS_KM.
DM_Rules	This type represents rules retrieved for Association Rules or *k*-means models using GET_ASSOCIATION_RULES and GET_MODEL_DETAILS_KM respectively.
DM_SVM_Attribute	This type represents each attribute in an attribute set for SVM model details
DM_SVM_Attribute_Set	This type represents a set of attributes returned by GET_MODEL_DETAILS_SVM for a linear model.
DM_SVM_Linear_Coeff	This type represents an entry in the set of linear coefficients returned by GET_MODEL_DETAILS_SVM
DM_SVM_Linear_Coeff_Set	This type represents the set of linear coefficients returned by GET_MODEL_DETAILS_SVM for an SVM model built using the linear kernel.

Table 4–9 through Table 4–16 list the constants to be used for various settings in the settings table.

Table 4–9 DBMS_DATA_MINING Constants Summary: Mining Function

Constant	Purpose
association	Parameter value for *mining_function* in CREATE_MODEL, representing association mining function
attribute importance	Parameter value for *mining_function* in CREATE_MODEL, representing attribute importance mining function
classification	Parameter value for *mining_function* in CREATE_MODEL, representing classification mining function

Table 4–9 (Cont.) DBMS_DATA_MINING Constants Summary: Mining Function

Constant	Purpose
`regression`	Parameter value for *`mining_function`* in `CREATE_MODEL`, representing regression mining function
`clustering`	Parameter value for *`mining_function`* in `CREATE_MODEL`, representing clustering mining function
`feature_extraction`	Parameter value for *`mining_function`* in `CREATE_MODEL`, representing Feature Extraction mining function

Table 4–10 DBMS_DATA_MINING Constants Summary: Function Settings

Constant	Purpose
`clas_priors_table_name`	Setting name representing prior probability table name for classification function
`clus_num_clusters`	Setting name representing number of clusters for clustering function
`feat_num_features`	Setting name representing number of features for feature selection function
`asso_max_rule_length`	Setting name representing maximum rule length
`asso_min_confidence`	Setting name representing minimum confidence
`asso_min_support`	Setting name representing minimum support

Table 4–11 DBMS_DATA_MINING Constants Summary: Algorithm Settings

Constant	Purpose
`algo_name`	Setting name representing the mining algorithm
`algo_apriori_association_rules`	Setting value for Apriori algorithm for association rules
`algo_naive_bayes`	Setting value for Naive Bayes (NB) algorithm for classification
`algo_support_vector_machines`	Setting value for Support Vector Machine (SVM) algorithm for classification or regression
`algo_nonnegative_matrix_factor`	Setting value for Non-Negative Matrix Factorization (NMF) for feature selection
`algo_kmeans`	Setting value for *k*-Means (KM) for clustering

Table 4–11 (Cont.) DBMS_DATA_MINING Constants Summary: Algorithm Settings

Constant	Purpose
algo_ai_mdl	Setting value for Minimum Description Length based algorithm for Attribute Importance.

Table 4–12 DBMS_DATA_MINING Constants Summary: Adaptive Bayes Network

Constant	Purpose
abns_model_type	Setting name representing ABN model type
abns_single_feature	Setting value representing single feature ABN model
abns_multi_feature	Setting value representing multi feature ABN model
abns_naive_bayes	Setting value representing Naive Bayes ABN model
abns_max_build_minutes	Setting name representing maximum time threshold to complete an ABN model build
abns_max_nb_predictors	Setting name representing the maximum number of Naive Bayes predictors to be considered for building an ABN model of type abns_naive_bayes

Table 4–13 DBMS_DATA_MINING Constants Summary: Naive Bayes

Constant	Purpose
nabs_singleton_threshold	Setting value for singleton threshold for Naive Bayes
nabs_pairwise_threshold	Setting value for pair-wise threshold for Naive Bayes

Table 4–14 DBMS_DATA_MINING Constants Summary: Support Vector Machines

Constant	Purpose
svms_kernel_function	Setting name representing the kernel function for SVM
svms_linear	Setting value for Linear Kernel for SVM
svms_guassian	Setting value for Gaussian Kernel for SVM
svms_kernel_cache_size	Setting name representing for Kernel Cache Size for Support Vector Machine
svms_conv_tolerance	Setting name representing tolerance for SVM
svms_std_dev	Setting name representing standard deviation for

Table 4–14 (Cont.) DBMS_DATA_MINING Constants Summary: Support Vector

Constant	Purpose
svms_complexity_factor	Setting name representing complexity factor for SVM
svms_epsilon	Setting name representing epsilon for SVM Regression

Table 4–15 DBMS_DATA_MINING Constants Summary: Non-Negative Matrix Factorization

Constant	Purpose
nmfs_num_iterations	Setting name representing number of iterations
nmfs_conv_tolerance	Setting name representing convergence tolerance
nmfs_random_seed	Setting name representing random seed for NMF

Table 4–16 DBMS_DATA_MINING Constants Summary: k-Means

Constant	Purpose
kmns_distance	Setting name representing distance function
kmns_euclidean	Setting value representing Euclidean distance function
kmns_cosine	Setting value representing cosine distance function
kmns_fast_cosine	Setting value representing fast cosine distance function
kmns_iterations	Setting name representing number of iterations
kmns_conv_tolerance	Setting name representing convergence tolerance
kmns_split_criterion	Setting name representing split criterion
kmns_variance	Setting value representing variance as the split criterion
kmns_size	Setting value representing size as the split criterion
kmns_block_growth	Setting name representing growth factor for memory allocated to hold cluster data
kmns_num_bins	Setting value for number of histogram bins
kmns_min_pct_attr_support	Setting value for minimum percentage report required for attributes in rules

Table 6–18 lists the errors generated by DBMS_DATA_MINING.

Table 4–17 DBMS DATA_MINING Errors Summary

Oracle Error	Description
ORA-40201	Invalid input parameter %s
ORA-40202	Column %s does not exist in the input table %s
ORA-40203	Model %s does not exist
ORA-40204	Model %s already exists
ORA-40205	Invalid setting name %s
ORA-40206	Invalid setting value for setting name %s
ORA-40207	Duplicate or multiple function settings
ORA-40208	Duplicate or multiple algorithm settings for function %s
ORA-40209	Invalid setting: %s for function %s
ORA-40211	Algorithm name: %s is invalid
ORA-40212	Invalid target data type in input data for function: %
ORA-40213	Contradictory values for settings: %s, %s
ORA-40214	Duplicate setting: %s
ORA-40215	Model %s is incompatible with current operation
ORA-40216	Feature not supported
ORA-40219	Apply result table %s is incompatible with current operation
ORA-40220	Maximum number of attributes exceeded
ORA-40221	Maximum target cardinality exceeded
ORA-40222	Data mining model export failed, job name=%s, error=%s
ORA-40223	Data mining model import failed, job name=%s, error=%s
ORA-40225	Model is currently in use by another process
ORA-40251	No support vectors were found
ORA-40252	No target values were found
ORA-40253	No target counter examples were found
ORA-40261	Input data for model build contains negative values
ORA-40262	NMF: number of features not between [1, %s]

Table 4–17 DBMS DATA_MINING Errors Summary

Oracle Error	Description
ORA-40271	No statistically significant features were found
ORA-40272	Apply rules prohibited for this model mode
ORA-40273	Invalid model type %s for Adaptive Bayes Network algorithm

Table 4–18 lists the user views provided by Oracle to obtain information about the models generated using DBMS_DATA_MINING.

Table 4–18 DBMS_DATA_MINING Summary of User Views

User View	Purpose
DM_USER_MODELS	Lists all models in a given user's schema.

4.8 Summary of DBMS_DATA_MINING Subprograms

Table 4–19 DBMS_DATA_MINING Summary of Subprograms

Data Type	Purpose
APPLY Procedure	Applies a model to scoring data
CREATE_MODEL Procedure	Creates (builds) a mining model
COMPUTE_CONFUSION_ MATRIX Procedure	Computes the confusion matrix from the APPLY results on test data for a classification model; also provides the accuracy of the model
COMPUTE_LIFT Procedure	Computes lift for a given positive target value from the APPLY results on test data for a classification model
COMPUTE_ROC Procedure	Computes Receiver Operating Characteristic for target attributes with binary class from the APPLY results on test data for a classification model.
DROP_MODEL Procedure	Drops a model
EXPORT_MODEL Procedure	Exports one or more models from a schema
GET_ASSOCIATION_RULES Function	This table function returns the rules from an Association model
GET_DEFAULT_SETTINGS Function	This table function returns all the default settings for all mining functions and algorithms.

Table 4–19 (Cont.) DBMS_DATA_MINING Summary of Subprograms

Data Type	Purpose
GET_FREQUENT_ITEMSETS Function	Returns a set of rows that represent the frequent itemsets from an Association model.
GET_MODEL_DETAILS_ABN Function	Provides the details of an Adaptive Bayes Network model
GET_MODEL_DETAILS_KM Function	Provides the details of a *k*-Means model
GET_MODEL_DETAILS_NB Function	Provides the details of a Naive Bayes model
GET_MODEL_DETAILS_NMF Function	Provides the details of an NMF model
GET_MODEL_DETAILS_SVM Function	Provides the details of a SVM model
GET_MODEL_SETTINGS Function	Provides the settings used to build a model
GET_MODEL_SIGNATURE Function	Provides the signature of a model
IMPORT_MODEL Procedure	Imports one or more models into the current schema
RANK_APPLY Procedure	Ranks the predictions from the APPLY results for a classification model
RENAME_MODEL Procedure	Renames a model

4.9 Model Export and Import

Data mining models can be moved between Oracle databases or schemas. For example, in an organization, data mining specialists may build and test data mining models in a data mining lab. After models are built and tested in the lab, the chosen model may be moved to a scoring engine used by applications. Because data mining lab and scoring engine usually do not share the same database, the model must be exported from the lab and then imported to the scoring engine. Model export and import can be a routine procedure. As new data are accumulated, data mining specialists will build and test new models, and newer and better models will be loaded onto the scoring engine on a regular basis. DBAs will want to back up and restore models in their routine database maintenance.

Native export and import of data mining models are supported in the following scenarios:

- Database export/import: When a DBA exports a full database using utility `expdp`, all the existing data mining models in the database are exported. By the same token, when a DBA imports a database dump using utility `impdp`, all the data mining models in the dump are restored.
- Schema export/import: When a user or DBA exports a schema using `expdp`, all the data mining models in the schema are exported. When the user or DBA imports the schema dump using `impdp`, all the models in the dump are imported.
- Selective model export/import: Users can export specific models using `DBMS_DATA_MINING.EXPORT_MODEL` and import specified models using `DBMS_DATA_MINING.IMPORT_MODEL`.

4.9.1 Limitations

The use of model export and import is subject to the following limitations:

- Only models built by the `DBMS_DATA_MINING` interface are supported.
- Only local export and import via dump files is supported. That is, models from a given local schema or database can be exported into a dump file. Models in a dump file can be imported into a local database or schema.

4.9.2 Prerequisites

Prerequisites for model export are as follows:

- A valid directory object must be made available to the operator who has write privileges. See `CREATE_DIRECTORY` in the *PL/SQL Packages and Types Reference*.
- The source must be an Oracle database with Data Mining option installed.
- For database and schema export, the new Data Pump utility `expdp` must be used.

Prerequisites for model import are as follows:

- There must be a valid directory object pointing to where dump files reside, and the operator must have read and write privileges.
- The destination database must have Data Mining option or Data Mining Scoring Engine option installed.
- Dump files must be created by `expdp` or `EXPORT_MODEL`.

- For database and schema import, the new Data Pump import utility `impdp` must be used

See also:

- *Data Pump Export, Oracle 10g Database Utilities, Part I.*
- *Data Pump Import, Oracle 10g Database Utilities, Part II.*

4.9.3 Choose the Right Utility

There are two ways to export models:

- Export all models, in a user schema or in the entire database.
- Export selected models in a user schema.

To export all data mining models in a user schema, you can either run `expdp` or use `EXPORT_MODEL` with the parameter `model_filter` set to `NULL`. Note the difference between the two operations: When you run `expdp` to export the schema, all objects in the schema including data mining models are exported. When you run `EXPORT_MODEL` with a `NULL` `model_filter`, only the models will be exported.

There are also two ways to import models from the dump file:

- Import all models as well as other database objects and data.
- Import models only, either all or a selected few from the dump file.

In general, if you want to import the full dump file set, run `impdp`. This imports all database objects and data, including all data mining models, from the dump file set. If you want to import models only, use `IMPORT_MODEL`. When the parameter `model_filter` is set to `NULL`, all models are imported from the dump. If valid model names are assigned in `model_filter`, this operation imports only named models from the dump file set.

4.9.4 Temp Tables

Data mining model export and import jobs will create and manage two temporary tables in the user schema: `DM$P_MODEL_EXPIMP_TEMP` and `DM$P_MODEL_TABKEY_TEMP`. Users should not manipulate these tables.

5

ODM PL/SQL Sample Programs

This chapter provides sample code using DBMS_DATA_MINING for all the supported algorithms. The dataset used is the Drug Depot dataset that is available as part of the sample schema in Oracle10*g*. Please refer to *Oracle Database Sample Schemas* for information on sample schemas.

All samples are available in the directory $ORACLE_HOME/dm/demo/sample/plsql.

ODM sample datasets need to be loaded into a user schema prior to using the sample programs. Refer to the following scripts for creating Oracle tablespace, user schema, and loading ODM sample datasets:

```
$ORACLE_HOME/dm/admin/odmtbs.sql
$ORACLE_HOME/dm/admin/odmuser.sql
$ORACLE_HOME/dm/admin/dmuserld.sql
$ORACLE_HOME/dm/admin/dmshgrants.sql
```

5.1 Overview of ODM PL/SQL Sample Programs

The ODM PL/SQL sample programs illustrate the main operations of the data mining process:

- Preparing the data
- Building a model
- Testing the model
- Applying the model to new data (scoring the data)

Data mining models can be either supervised or unsupervised.

Supervised models predict the value of a specified variable, called the target variable, together with the confidence associated with each prediction. Supervised

models are illustrated in the sample programs for Naive Bayes (NB), Adaptive Bayes Networks (ABN), and Support Vector Machines (SVM).

Unsupervised models have no target variable; they are used to predict group membership or relationships of an individual. Unsupervised models are illustrated in the sample programs for Clustering, Association Rules, and Non-Negative Matrix Factorization. Attribute Importance is also illustrated.

The PL/SQL sample programs rely on two sets of data:

- Individual datasets: All samples named *algorithm*_demo.sql are based on these datasets. These datasets must be loaded using $ORACLE_HOME/dm/admin/dmuserld.sql in the user schema executing these demos.
- SH schema dataset: All samples named *algorithm*_sh.sql are based on datasets derived from the SH schema. The SH schema must be installed as part of RDBMS installation. The script $ORACLE_HOME/dm/admin/dmshgrants.sql must be run by a user with privileges to access the SH schema, and the script $ORACLE_HOME/dm/admin/dmsh.sql must be run in the user schema executing these demos.

The file $ORACLE_HOME/dm/demo/data/README.txt explains the datasets.

Each sample program for demonstrating Classification (NB, ABN, SVM) contains code that prepares the input data using DBMS_DATA_MINING_TRANSFORM, builds a model, tests a model, and then scores the model against new data. It demonstrates how to generate test results such as a confusion matrix, lift, ROC, and ranked Apply results.

The samples for Regression using SVM normalize the input data, build models, and test models using metrics such as root mean squared error, apply the models to new data, and generate ranked results.

The samples for Association demonstrate model build, and show how to obtain frequent itemsets and association rules for a given support and confidence.

The samples for Clustering demonstrate model build, and show how to obtain clustering details such as histograms, child nodes, and rules. The clusters are scored and ranked based on their probability.

The samples for Feature Extraction demonstrate model build, and show how to obtain details of various features. The features are scored and ranked based on their probability.

There is one sample program demonstrating the BLAST interface for biological sequence match and alignment.

Finally, there are three sample programs that demonstrate text mining for extracting features from a text document into a nested table column, text classification using SVM, and text feature extraction using NMF, respectively.

5.2 Summary of ODM PL/SQL Sample Programs

All the sample programs listed in the tables below are located in the directory `$ORACLE_HOME/dm/demo/sample/plsql`.

The summary description of these sample programs is also provided in `$ORACLE_HOME/dm/demo/sample/plsql/README.txt`.

Table 5–1 PL/SQL Samples Based on Individual Datasets

Sample Program	Description
`aidemo.sql`	Attribute Importance using an MDL-based algorithm.
`abndemo.sql`	Classification using Adaptive Bayes Network algorithm
`ardemo.sql`	Association using Apriori algorithm
`blastdemo.sql`	BLAST sequence matching and alignment
`kmdemo.sql`	Clustering using *k*-Means algorithm
`nbdemo.sql`	Classification using Naive Bayes algorithm
`nmfdemo.sql`	Feature Extraction using NMF algorithm
`svmcdemo.sql`	Classification using SVM algorithm
`svmrdemo.sql`	Regression using SVM algorithm

Table 5–2 PL/SQL Samples Based on SH Schema

Sample Program	Description
`ai_sh.sql`	Attribute Importance using an MDL-based algorithm
`abn_sh.sql`	Classification using Adaptive Bayes Network algorithm
`ar_sh_.sql`	Association using Apriori algorithm
`akm_sh.sql`	Clustering using *k*-Means algorithm
`nb_sh.sql`	Classification using Naive Bayes algorithm
`nmf_sh.sql`	Feature Extraction using NMF algorithm

Table 5–2 (Cont.) PL/SQL Samples Based on SH Schema

Sample Program	Description
`svmc_sh.sql`	Classification using SVM algorithm
`svmr_sh.sql`	Regression using SVM algorithm
`textfe.sql`	Demonstrates extracting text features from a `CLOB/VARCHAR2` column into a nested table column in a table that can be provided as input to `CREATE_MODEL`
`textnmf.sql`	Text feature extraction using NMF
`textsvmc.sql`	Text classification using SVM

6

Sequence Matching and Annotation (BLAST)

This chapter describes table functions included with ODM that permit you to perform similarity searches against nucleotide and amino acid sequence data stored in an Oracle database. You can use the table functions described in this chapter for ad hoc searches or you can embed them in applications. The inclusion of these table functions in ODM positions Oracle as a platform for bioinformatics.

This chapter discusses the following topics:

- NCBI BLAST
- Using ODM BLAST

6.1 NCBI BLAST

The National Center for Biotechnology Information (NCBI) implemented one of the commonly used versions of the Basic Local Alignment Search Tool (BLAST).

Sequence alignments provide a way to compare new sequences with previously characterized sequences. Both functional and evolutionary information can be inferred from well-designed queries and alignments. BLAST provides a method for searching of both nucleotide and protein databases. Since the BLAST algorithm detects local alignments, regions of similarity embedded in otherwise unrelated sequences can be detected.

The BLAST algorithm searches nucleotide and amino acid query sequences against databases of nucleotide and amino acid sequences. Based on the nature of the query and the database sequences, NCBI BLAST provides the following variants:

- BLASTP compares an amino acid query sequence against an amino acid sequence database.

- BLASTN compares a nucleotide query sequence against a nucleotide sequence database.
- BLASTX compares a nucleotide query sequence translated along all six reading frames (both strands) against a amino acid sequence database.
- TBLASTN compares an amino acid query sequence against a nucleotide sequence database translated along all six reading frames (both strands).
- TBLASTX compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.

For more information about NCBI BLAST, see the NCBI BLAST Home Page at `http://www.ncbi.nlm.nih.gov/BLAST/`.

The table functions described in this chapter implement several of the variants of NCBI BLAST version 2.0.

6.2 Using ODM BLAST

This section contains several examples of using the ODM BLAST table functions to perform searches on nucleotide or amino acid sequences.

Most table function parameters have defaults. The defaults were carefully chosen so that users who have limited experience with BLAST will obtain good results.

6.2.1 Using BLASTN_MATCH to Search DNA Sequences

The BLAST table functions accept the CLOB (Character Large OBject) data type as the query sequence. It is not possible to construct a CLOB in an ad hoc SQL query. One way to construct a CLOB is to create a table and insert the query sequence into the table. Another option is to construct a CLOB using the programmatic interface if the BLAST query is part of a larger program. Suppose that the table `query_db` has the schema `(sequence_id VARCHAR2(32), sequence CLOB)`. The following SQL query inserts the query sequence into `query_db`:

```
INSERT INTO query_db VALUES ('1', 'AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGT');
```

Suppose that the table GENE_DB stores DNA sequences. Suppose that GENE_DB has attributes `seq_id`, `publication date`, `modification date`, `organism`, and `sequence`, among other attributes. There is no required schema for the table that stores the sequences. The only requirement is that the table contain an identifier and the sequence and any number of other optional attributes.

The portion of the database to be used for the search can be specified using SQL. The full power of SQL can be used to perform more sophisticated selections.

6.2.1.1 Searching for Good Matches in DNA Sequences

The following query does a BLAST search of the given query sequence against the human genome and returns the `seq_id`, `score`, and `expect` value of matches that score > 25:

```
SELECT t.t_seq_id, t.score, t.expect
FROM TABLE (
    BLASTN_MATCH (
       (SELECT sequence FROM query_db WHERE sequence_id = '1'),
       CURSOR (SELECT seq_id, sequence FROM GENE_DB
        WHERE organism = 'human'),
       1,
       -1,
       0,
       0,
       10,
       0,
       0,
       0,
       0,
       11,
       0,
       0)
) t WHERE t.score > 25;
```

Note: The parameter value of 0 invokes the default values in most cases. See the syntax for details.

6.2.1.2 Searching DNA Sequences Published After a Certain Date

The following query does the BLAST search against all sequences published after Jan 01, 2000:

```
SELECT t.t_seq_id, t.score, t.expect
FROM TABLE (
    BLASTN_MATCH (
       (SELECT sequence FROM query_db WHERE sequence_id = '1'),
       CURSOR (SELECT seq_id, sequence FROM GENE_DB
        WHERE publication_date > '01-JAN-2000'),
       1,
       -1,
```

```
        0,
        0,
        10,
        0,
        0,
        0,
        0,
        11,
        0,
        0)
) t WHERE t.score > 25;
```

You can obtain other attributes of the matching sequence by joining the BLAST result with the original sequence table as follows:

```
SELECT t.t_seq_id, t.score, t.expect, g.publication_date, g.organism
FROM GENE_DB g, TABLE (
    BLASTN_MATCH (
       (SELECT sequence FROM query_db WHERE sequence_id = '1'),
       CURSOR (SELECT seq_id, sequence FROM GENE_DB
        WHERE publication_date > '01-JAN-2000'),
       1,
       -1,
       0,
       0,
       10,
       0,
       0,
       0,
       0,
       11,
       0,
       0)
) t WHERE t.t_seq_id = g.seq_id AND t.score > 25;
```

6.2.2 Using BLASTP_MATCH to Search Protein Sequences

Suppose that the table PROT_DB stores protein sequences. Insert the protein query sequence to be used for the search into `query_db`.

6.2.2.1 Searching for Good Matches in Protein Sequences

The following query does a BLASTP search of the given query sequence against protein sequences in PROT_DB and returns the `identifier`, `score`, `name`, and `expect` value of matches that score > 25:

```
SELECT t.t_seq_id, t.score, t.expect, p.name
FROM PROT_DB p, TABLE(
       BLASTP_MATCH (
         (SELECT sequence FROM query_db WHERE sequence_id = '2'),
         CURSOR(SELECT seq_id, sequence FROM PROT_DB),
         1,
         -1,
         0,
         0,
         'BLOSUM62',
         10,
         0,
         0,
         0,
         0,
         0)
       )t WHERE t.t_seq_id = p.seq_id AND t.score > 25
          ORDER BY t.expect;
```

6.2.3 Using BLASTN_ALIGN to Search and Align DNA Sequences

Suppose that the table GENE_DB stores DNA sequences. Suppose that GENE_DB has attributes `seq_id`, `publication date`, `modification date`, `organism`, and `sequence` among other attributes.

6.2.3.1 Searching and Aligning for Good Matches in DNA Sequences

The following query does a BLAST search and alignment of the given query sequence against the human genes and returns the `publication_date`, organism, and the alignment attributes of matching sequences that score > 25 and where more than 50% of the sequence is conserved in the match:

```
SELECT t.t_seq_id, t.alignment_length, t.pct_identity,
       t.q_seq_start, t.q_seq_end, t.t_seq_start, t.t_seq_end,
       t.score, t.expect, g.publication_date, g.organism
FROM GENE_DB g, TABLE (
    BLASTN_ALIGN (
       (SELECT sequence FROM query_db WHERE sequence_id = '1'),
       CURSOR (SELECT seq_id, sequence FROM GENE_DB
        WHERE publication_date > '01-JAN-2000'),
       1,
       -1,
       0,
       0,
       10,
```

```
       0,
       0,
       0,
       0,
       11,
       0,
       0)
) t WHERE t.t_seq_id = g.seq_id AND t.score > 25
    AND t.pct_identity > 50;
```

You can use BLASTP_ALIGN and TBLAST_ALIGN in a similar way.

6.2.4 Output of the Table Function

The output of a BLAST query is a table; the output table is described as the output table for the specific query.

Here are two examples of queries and the resulting output tables.

Query 1 is as follows:

```
select T_SEQ_ID AS seq_id, score, EXPECT as evalue
  from TABLE(
       BLASTP_MATCH (
         (select sequence from query_db),
         CURSOR(SELECT seq_id, seq_data
                FROM swissprot
                WHERE organism = 'Homo sapiens (Human)'),
         1,
         -1,
         0,
         0,
         'BLOSUM62',
         10,
         0,
         0,
         0,
         0,
         0)
       );
```

The output for query 1 is as follows:

```
SEQ_ID        SCORE     EVALUE
-------- ----------     ----------
P31946          205     5.8977E-18
Q04917          198     3.8228E-17
P31947          169     8.8130E-14
P27348          198     3.8228E-17
P58107           49     7.24297332
```

Query 2 is as follows:

```
select T_SEQ_ID AS seq_id, ALIGNMENT_LENGTH as len,
       Q_SEQ_START as q_strt, Q_SEQ_END as q_end, Q_FRAME, T_SEQ_START as t_
strt,
       T_SEQ_END as t_end, T_FRAME, score, EXPECT as evalue
  from TABLE(
       BLASTP_ALIGN (
         (select sequence from query_db),
         CURSOR(SELECT seq_id, seq_data
                FROM swissprot
                WHERE organism = 'Homo sapiens (Human)' AND
                      creation_date > '01-Jan-90'),
         1,
         -1,
         0,
         0,
         'BLOSUM62',
         10,
         0,
         0,
         0,
         0,
         0)
       );
```

The output for Query 2 is as follows:

```
SEQ_ID    LEN Q_STRT Q_END Q_FRAME T_STRT T_END T_FRAME   SCORE  EVALUE
-------- ---- ------ ----- ------- ------ ----- ------- ------- ----------
P31946     50      0    50       0     13    63       0     205  5.1694E-18
Q04917     50      0    50       0     12    62       0     198  3.3507E-17
P31947     50      0    50       0     12    62       0     169  7.7247E-14
P27348     50      0    50       0     12    62       0     198  3.3507E-17
P58107     21     30    51       0    792   813       0      49  6.34857645
```

6.2.5 Sample Data for BLAST

We provide a few sample datasets and queries to test if the BLAST functions work correctly after ODM is installed.

The DM_USER schema contains the following sequence data tables.

SWISSPROT: This table contains the sequences in Release 40 of the SwissProt dataset. This table has the sequence identifier, creation_date, organism, and sequence_data attributes. It has 101,602 protein sequences.

```
SQL> describe SWISSPROT;
Name                                      Null?    Type
----------------------------------------- -------  -------------
SEQ_ID                                             VARCHAR2(32)
CREATION_DATE                                      DATE
ORGANISM                                           VARCHAR2(256)
SEQ_DATA                                           CLOB
```

PROT_DB: This table consists of 19 protein sequences from the SwissProt dataset.

```
SQL> describe prot_db;
Name                                       Null?    Type
----------------------------------------- -------  -------------
SEQ_ID                                             VARCHAR2(32)
SEQ_DATA                                           CLOB
```

ECOLI10: This table contains 10 nucleotide sequences from the ecoli dataset.

```
SQL> describe ECOLI10;
Name                                        Null?    Type
----------------------------------------- -------- ---------------
    SEQ_ID                                         VARCHAR2(32)
    SEQ_DATA                                       CLOB
```

Table 6–1 displays genetic codes and associated names.

Table 6–1 Table of Genetic Codes

Genetic Code	Name
1	Standard
2	Vertebrate Mitochondrial

Table 6–1 (Cont.) Table of Genetic Codes

Genetic Code	Name
3	Yeast Mitochondrial
4	Mold Mitochondrial, Protozoan Mitochondrial, Coelenterate Mitochondrial, Mycoplasma, Spiroplasm
5	Invertebrate Mitochondrial
6	Ciliate Nuclear, Dasycladacean Nuclear, Hexamita Nuclear
9	Echinoderm Mitochondrial
10	Euplotid Nuclear
11	Bacterial and Plant Plastid
12	Alternative Yeast Nuclear
13	Ascidian Mitochondrial
14	Flatworm Mitochondrial
15	Blepharisma Macronuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial Code

There are several public domain sequence databases available. One of them is the SwissProt database, which is a highly curated collection of protein sequences. The SwissProt database can be downloaded from

```
ftp://ftp.ebi.ac.uk/pub/databases/swissprot/release/sprot40.dat
```

In addition to the raw sequence data, the SwissProt database contains several other attributes of the sequence including organism, date published, date modified, published literature references, annotations, etc. BLAST requires only the sequence identifier and the sequence data to be stored to perform searches.

Depending on the needs of your specific application, different sets of these attributes may be important. Therefore, the database schema required to store the data needs to be appropriately designed. You can use a scripting language to parse the required fields from the SwissProt data and format the fields so that they can be loaded into an Oracle database.

The following Perl script outputs the sequence identifier, creation_date, organism, and sequence data in the required format for `sqlldr` is given below. (`sqlldr` is a program to load data into an Oracle database.)

```
#!/bin/perl
#swissprot.pl < input > output
#Input: protein db as provided by SWISSPROT
#
my $string = "";
my $indicator = "";
$sq = 0;
$ac = 0;

while(<>)
{
    #chop;
    if ( /^\/\// ) {
      print "\n";
      $sq = 0;
      $ac = 0;
      next;
    }
    if ($sq == 1) {
        @words = split;
        foreach $word (@words) {
          print "$word";
        }
        next;
    }
    if( /^AC(\s+)(\w+);/ ) {
      if ($ac == 0) {
        $indicator = $2;
        print "$indicator|";
        $sq = 0;
        $dt = 0;
        $ac = 1;
        next;
      }
    }
    if ( /^OS(\s+)(.*)\./ ) {
        $organism = $2;
        print "$organism|";
        next;
    }
    if ( /^DT(\s+)(\S+)/ ) {
```

```
        if ($dt == 0) {
           print "$2|";
           $dt = 1;
        }
    }
    if ( /^SQ(\s+)/ ) {
        $sq = "1";
        next;
    }
}
```

Follow these steps to download, parse, and save the SwissProt data in an Oracle database:

1. Download SwisProt data to the file `sprot40.dat`.
2. Save the perl script in a file named `swissprot.pl`, type the command

   ```
   swissprot.pl sprot40.dat > sprot_formatted.txt
   ```

   This command will read the SwissProt data stored in `sprot40.dat`, format it, and write it out to `sprot_formatted.txt`.
3. In order to load the data using `sqlldr`, you must create a table to hold the data and a control file. Create the table `swissprot` using the following SQL statement:

   ```
   create table swissprot (SEQ_ID VARCHAR2(32), CREATION_DATE DATE,
   ORGANISM VARCHAR2(256), SEQ_DATA CLOB);
   ```

   Create a control file named `sprot.ctl` with the following contents:

   ```
   LOAD DATA
   INFILE sprot40_formatted.txt
   INTO TABLE swissprot
   REPLACE
   FIELDS TERMINATED BY '|'
   TRAILING NULLCOLS
   (
   seq_id,
   creation_date,
   organism,
   seq_data char(100000)
   )
   ```

4. Finally, use the following command to load the data:

```
sqlldr userid=<user_name>/<passwd> control=sprot.ctl log=sprot.log
direct=TRUE data=sprot40_formatted.txt
```

The SwisProt data is now stored in the Oracle table `swissprot`.

Summary of BLAST Table Functions

The BLAST functionality is available as built-in table functions; these table functions can be used in the FROM clause of a SQL query.

Table 6–2 BLAST Table Functions

Table Function	Description
BLASTN_MATCH Table Function	Perform a BLASTN search of the given nucleotide sequence against the selected portion of the nucleotide database
BLASTP_MATCH Table Function on page 6-17	Perform a BLASTP search of the given amino acid sequence against the selected portion of the protein database
TBLAST_MATCH Table Function on page 6-20	Perform BLAST searches involving translations of either the query sequence or the database of sequences
BLASTN_ALIGN Table Function on page 6-23	Performs a BLASTN alignment of the given nucleotide sequence against the selected portion of the nucleotide database
BLASTP_ALIGN Table Function on page 6-27	Perform a BLASTP alignment of the given amino acid sequence against the selected portion of the protein database
TBLAST_ALIGN Table Function on page 6-30	Perform BLAST alignments involving translations of either the query sequence or the database of sequences

BLASTN_MATCH Table Function

This table function performs a BLASTN search of the given nucleotide sequence against the selected portion of the nucleotide database. The database can be selected using a standard SQL select and passed into the function as a cursor. It accepts the standard BLAST parameters that are mentioned below. The match returns the identifier of the matched (target) sequence (`t_seq_id`) (for example, the NCBI accession number), the score of the match, and the expect value.

Syntax

```
function BLASTN_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 5,
  extend_gap_cost NUMBER default 2,
  mismatch_cost NUMBER default -3,
  match_reward NUMBER default 1,
  word_size NUMBER default 11,
  xdropoff NUMBER default 30,
  final_x_dropoff NUMBER default 50)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parameters

Table 6–3 describes the input parameters for BLASTN_MATCH; Table 6–4, the output parameters.

Table 6–3 Input Parameters for BLASTN_MATCH Table Function

Parameter	Description
`query_seq`	The query sequence to search. This version of ODM BLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
`seqdb_cursor`	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.

Table 6–3 Input Parameters for BLASTN_MATCH Table Function

Parameter	Description
subsequence_from	Start position of a region of the query sequence to be used for the search. The default is 1.
subsequence_to	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as subsequence_to. The default is -1.
filter_low_complexity	TRUE or FALSE. If TRUE, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence. The default is FALSE.
mask_lower_case	TRUE or FALSE. If TRUE, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence. The default is FALSE.
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.
open_gap_cost	The cost of opening a gap. The default value is 5. Specifying 0 invokes default behavior.
extend_gap_cost	The cost of extending a gap. The default value is 2. Specifying 0 invokes default behavior.
mismatch_cost	The penalty for nucleotide mismatch. The default value is -3. Specifying 0 invokes default behavior.
match_reward	The reward for a nucleotide match. The default value is 1. Specifying 0 invokes default behavior.
word_size	The word size used for dividing the query sequence into subsequences during the search. The default value is 11. Specifying 0 invokes default behavior.
xdropoff	Dropoff for BLAST extensions in bits. The default value is 30. Specifying 0 invokes default behavior.
final_x_dropoff	The final X dropoff value for gapped alignments in bits. The default value is 50. Specifying 0 invokes default behavior.

Table 6–4 Output Parameters for BLASTN_MATCH Table Function

Attribute	Description
`t_seq_id`	The sequence identifier of the returned match.
`score`	The score of the returned match.
`expect`	The expect value of the returned match.

BLASTP_MATCH Table Function

This table function performs a BLASTP search of the given amino acid sequence against the portion of the selected protein database. The database can be selected using a standard SQL select and passed into the function as a cursor. We also accept the standard BLAST parameters that are mentioned below. The match returns the identifier of the matched (target) sequence (t_seq_id) (for example, the NCBI accession number), the score of the match, and the expect value.

Syntax

```
function BLASTP_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parameters

Table 6–5 describes the input parameters for BLASTN_MATCH; Table 6–6, the output parameters.

Table 6–5 Input Parameters for BLASTP_MATCH Table Function

Parameter	Description
query_seq	The query sequence to search. This version of ODM BLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
seqdb_cursor	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.

Table 6–5 Input Parameters for BLASTP_MATCH Table Function

Parameter	Description
subsequence_from	Start position of a region of the query sequence to be used for the search. The default is 1.
subsequence_to	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as subsequence_to. The default is -1.
filter_low_complexity	TRUE or FALSE. If TRUE, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence. The default value is FALSE.
mask_lower_case	TRUE or FALSE. If TRUE, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence. The default value is FALSE.
sub_matrix	Specifies the substitution matrix used to assign a score for aligning any possible pair of residues. The different options are PAM30, PAM70, BLOSUM80, BLOSUM62, and BLOSUM45. The default is BLOSUM62.
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.
open_gap_cost	The cost of opening a gap. The default value is 11. Specifying 0 invokes default behavior.
extend_gap_cost	The cost of extending a gap. The default value is 1. Specifying 0 invokes default behavior.
word_size	The word size used for dividing the query sequence into subsequences during the search. The default value is 3. Specifying 0 invokes default behavior.
x_dropoff	Dropoff for BLAST extensions in bits. The default value is 15. Specifying 0 invokes default behavior.
final_x_dropoff	The final X dropoff value for gapped alignments in bits. The default value is 25. Specifying 0 invokes default behavior.

Table 6–6 Output Parameters for BLASTP_MATCH Table Function

Attribute	Description
t_seq_id	The sequence identifier of the returned match.
score	The score of the returned match.
expect	The expect value of the returned match.

TBLAST_MATCH Table Function

This table function performs BLAST searches involving translations of either the query sequence or the database of sequences. The available options are:

- BLASTX: The query nucleotide sequence is translated and compared against a protein database.
- TBLASTN: The query amino acid sequence is compared against a translated nucleotide database.
- TBLASTX: The query nucleotide sequence is translated and compared against a translated nucleotide database.

The database can be selected using a standard SQL select and passed into the function as a cursor. We also accept the standard BLAST parameters that are mentioned below. The match returns the identifier of the matched (target) sequence (t_seq_id) (for example, the NCBI accession number), the score of the match, and the expect value.

Syntax

```
function TBLAST_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  translation_type VARCHAR2 default 'BLASTX',
  genetic_code NUMBER default 1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parameters

Table 6–7 describes the input parameters for TBLAST_MATCH; Table 6–8, the output parameters.

Table 6–7 Input Parameters for TBLAST_MATCH Table Function

Parameter	Description
query_seq	The query sequence to search. This version of ODM BKLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
seqdb_cursor	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.
subsequence_from	Start position of a region of the query sequence to be used for the search. The default is 1.
subsequence_to	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as subsequence_to. The default is -1.
translation_type	Type of the translation involved. The options are BLASTX, TBLASTN, and TBLASTX. The default is BLASTX.
genetic_code	Used for translating nucleotide sequences to amino acid sequences. genetic_code is sort of like a mapping table. NCBI supports 17 different genetic codes. The supported genetic codes and their names are given in Table 6–1. The default genetic code is 1.
filter_low_complexity	TRUE or FALSE. If TRUE, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence. The default is FALSE.
mask_lower_case	TRUE or FALSE. If TRUE, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence. The default is FALSE.
sub_matrix	Specifies the substitution matrix used to assign a score for aligning any possible pair of residues. The different options are PAM30, PAM70, BLOSUM80, BLOSUM62, and BLOSUM45. The default is BLOSUM62.

Table 6–7 Input Parameters for TBLAST_MATCH Table Function

Parameter	Description
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.
open_gap_cost	The cost of opening a gap. The default value is 11. Specifying 0 invokes default behavior.
extend_gap_cost	The cost of extending a gap. The default value is 1. Specifying 0 invokes default behavior.
word_size	The word size used for dividing the query sequence into subsequences during the search. The default value is 3. Specifying 0 invokes default behavior.
x_dropoff	Dropoff for BLAST extensions in bits. The default value is 15. Specifying 0 invokes default behavior.
final_x_dropoff	The final X dropoff value for gapped alignments in bits. The default value is 25. Specifying 0 invokes default behavior.

Table 6–8 Output Parameters for TBLAST_MATCH Table Function

Attribute	Description
t_seq_id	The sequence identifier of the returned match.
score	The score of the returned match.
expect	The expect value of the returned match.

BLASTN_ALIGN Table Function

This table function performs a BLASTN alignment of the given nucleotide sequence against the selected portion of the nucleotide database. The database can be selected using a standard SQL select and passed into the function as a cursor. It accepts the standard BLAST parameters that are mentioned below.

BLASTN_MATCH returns only the score and expect value of the match. It does not return information about the alignment. BLASTN_MATCH is typically used when you want to follow up a BLAST search with a Smith-Waterman alignment.

BLASTN_ALIGN does the BLAST alignment and returns the information about the alignment.

Syntax

```
function BLASTN_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 5,
  extend_gap_cost NUMBER default 2,
  mismatch_cost NUMBER default -3,
  match_reward NUMBER default 1,
  word_size NUMBER default 11,
  xdropoff NUMBER default 30,
  final_x_dropoff NUMBER default 50)
  return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
    t_seq_end NUMBER,
```

```
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parameters

Table 6–9 describes the input parameters for BLASTN_ALIGN; Table 6–10, the output parameters.

Table 6–9 Input Parameters for BLASTN_ALIGN Table Function

Parameter	Description
query_seq	The query sequence to search. This version of ODM BLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
seqdb_cursor	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.
subsequence_from	Start position of a region of the query sequence to be used for the search. The default is 1.
subsequence_to	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as subsequence_to. The default is -1.
filter_low_complexity	TRUE or FALSE. If TRUE, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence.
mask_lower_case	TRUE or FALSE. If TRUE, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence. The default is FALSE.
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.
open_gap_cost	The cost of opening a gap. The default value is 5. Specifying 0 invokes default behavior.
extend_gap_cost	The cost of extending a gap. The default value is 2. Specifying 0 invokes default behavior.

Table 6–9 Input Parameters for BLASTN_ALIGN Table Function

Parameter	Description
mismatch_cost	The penalty for nucleotide mismatch. The default value is -3. Specifying 0 invokes default behavior.
match_reward	The reward for a nucleotide match. The default value is 1. Specifying 0 invokes default behavior.
word_size	The word size used for dividing the query sequence into subsequences during the search. The default value is 11. Specifying 0 invokes default behavior.
xdropoff	Dropoff for BLAST extensions in bits. The default value is 30. Specifying 0 invokes default behavior.
final_x_dropoff	The final X dropoff value for gapped alignments in bits. The default value is 50. Specifying 0 invokes default behavior.

Table 6–10 Output Parameters for BLASTN_ALIGN Table Function

Parameter	Description
t_seq_id	Identifier (for example, the NCBI accession number) of the matched (target) sequence
pct_identity	Percentage of the query sequence that identically matches with the database sequence.
alignment_length	Length of the alignment.
mismatches	Number of base-pair mismatches between the query and the database sequence.
positives	Number of base-pairs with a positive match score between the query and the database sequence.
gap_openings	Number of gaps opened in gapped alignment.
gap_list	List of offsets where a gap is opened.
q_seq_start, q_seq_end	The indices of the portion of the query sequence that is aligned
q_frame	Translation frame number of the query.
t_seq_start, t_seq_end	The indices of the portion of the target sequence that is aligned.
t_frame	Translation frame number of the target sequence.
expect	Expect value of the alignment.

Table 6–10 Output Parameters for BLASTN_ALIGN Table Function

Parameter	Description
`score`	Score corresponding to the alignment.

BLASTP_ALIGN Table Function

This table function performs a BLASTP alignment of the given amino acid sequences against the selected portion of the protein database. The database can be selected using a standard SQL select and passed into the function as a cursor. You can also use the standard BLAST parameters that are mentioned below.

BLASTP_MATCH function returns only the score and expect value of the match. It does not return information about the alignment. The BLASTP_MATCH function will typically be used where the user wants to follow up a BLAST search with a full FASTA or Smith-Waterman alignment.

The BLASTP_ALIGN function does the BLAST alignment and returns the information about the alignment. The schema of the returned alignment is the same as that of BLASTN_ALIGN.

Syntax

```
function SYS_BLASTP_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
 return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
```

```
    t_seq_end NUMBER,
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parameters

Table 6–11 describes the input parameters for BLASTP_ALIGN; Table 6–12, the output parameters.

Table 6–11 Input Parameters for BLASTP_ALIGN Table Function

Parameter	Description
query_seq	The query sequence to search. This version of ODM BKLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
seqdb_cursor	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.
subsequence_from	Start position of a region of the query sequence to be used for the search. The default is 1.
subsequence_to	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as subsequence_to. The default is -1.
filter_low_complexity	TRUE or FALSE. If TRUE, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence. The default is FALSE.
mask_lower_case	TRUE or FALSE. If TRUE, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence. The default is FALSE.
sub_matrix	Specifies the substitution matrix used to assign a score for aligning any possible pair of residues. The different options are PAM30, PAM70, BLOSUM80, BLOSUM62, and BLOSUM45. The default is BLOSUM62.
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.

Table 6–11 Input Parameters for BLASTP_ALIGN Table Function

Parameter	Description
`open_gap_cost`	The cost of opening a gap. The default value is 11. Specifying 0 invokes default behavior.
`extend_gap_cost`	The cost of extending a gap. The default value is 1. Specifying 0 invokes default behavior.
`word_size`	The word size used for dividing the query sequence into subsequences during the search. The default value is 3. Specifying 0 invokes default behavior.
`x_dropoff`	X-dropoff for BLAST extensions in bits. The default value is 15. Specifying 0 invokes default behavior.
`final_x_dropoff`	The final X dropoff value for gapped alignments in bits. The default value is 25 Specifying 0 invokes default behavior.

Table 6–12 Output Parameters for BLASTP_ALIGN Table Function

Parameter	Description
`t_seq_id`	Identifier (for example, the NCBI accession number) of the matched (target) sequence
`pct_identity`	Percentage of the query sequence that identically matches with the database sequence.
`alignment_length`	Length of the alignment.
`mismatches`	Number of base-pair mismatches between the query and the database sequence.
`positives`	Number of base-pairs with a positive match score between the query and the database sequence.
`gap_openings`	Number of gaps opened in gapped alignment.
`gap_list`	List of offsets where a gap is opened.
`q_seq_start`, `q_seq_end`	The indices of the portion of the query sequence that is aligned.
`q_frame`	Translation frame number of the query.
`t_seq_start`, `t_seq_end`	The indices of the portion of the target sequence that is aligned.
`t_frame`	Translation frame number of the target sequence.
`score`	Score corresponding to the alignment.

TBLAST_ALIGN Table Function

This table function performs BLAST alignments involving translations of either the query sequence or the database of sequences. The available translation options are BLASTX, TBLASTN, and TBLASTX. The schema of the returned alignment is the same as that of BLASTN_ALIGN and BLASTP_ALIGN.

Syntax

```
function TBLAST_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default 0,
  translation_type VARCHAR2 default 'BLASTX',
  genetic_code NUMBER default 1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
 return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
    t_seq_end NUMBER,
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parameters

Table 6–13 describes the input parameters for TBLAST_ALIGN; Table 6–14, the output parameters.

Table 6–13 Input Parameters for TBLAST_ALIGN Table Function

Parameter	Description
`query_seq`	The query sequence to search. This version of ODM BKLAST accepts bare sequences only. A *bare sequence* is just lines of sequence data. Blank lines are not allowed in the middle of bare sequence input.
`seqdb_cursor`	The cursor parameter supplied by the user when calling the function. It should return two columns in its returning row, the sequence identifier and the sequence string.
`subsequence_from`	Start position of a region of the query sequence to be used for the search. The default is 1.
`subsequence_to`	End position of a region of the query sequence to be used for the search. If -1 is specified, the sequence length is taken as `subsequence_to`. The default is -1.
`translation_type`	Type of the translation involved. The options are BLASTX, TBLASTN, and TBLASTX. The default is BLASTX.
`genetic_code`	Used for translating nucleotide sequences to amino acid sequences. `genetic_code` is sort of like a mapping table. NCBI supports 17 different genetic codes. The supported genetic codes and their names are given in Table 6–1. The default genetic code is 1.
`filter_low_ complexity`	`TRUE` or `FALSE`. If `TRUE`, the search masks off segments of the query sequence that have low compositional complexity. Filtering can eliminate statistically significant but biologically uninteresting regions, leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence.The default is `FALSE`.
`mask_lower_case`	`TRUE` or `FALSE`. If `TRUE`, you can specify a sequence in upper case characters as the query sequence and denote areas to be filtered out with lower case. This customizes what is filtered from the sequence.The default is `FALSE`.
`sub_matrix`	Specifies the substitution matrix used to assign a score for aligning any possible pair of residues. The different options are `PAM30`, `PAM70`, `BLOSUM80`, `BLOSUM62`, and `BLOSUM45`. The default is `BLOSUM62`.

Table 6–13 Input Parameters for TBLAST_ALIGN Table Function

Parameter	Description
expect_value	The statistical significance threshold for reporting matches against database sequences. The default value is 10. Specifying 0 invokes default behavior.
open_gap_cost	The cost of opening a gap. The default value is 11. Specifying 0 invokes default behavior.
extend_gap_cost	The cost of extending a gap. The default value is 1. Specifying 0 invokes default behavior.
word_size	The word size used for dividing the query sequence into subsequences during the search. The default value is 3. Specifying 0 invokes default behavior.
x_dropoff	Dropoff for BLAST extensions in bits. The default value is 15. Specifying 0 invokes default behavior.
final_x_dropoff	The final X dropoff value for gapped alignments in bits. The default value is 25. Specifying 0 invokes default behavior.

Table 6–14 Output Parameters for TBLAST_ALIGN Table Function

Parameter	Description
t_seq_id	Identifier (for example, the NCBI accession number) of the matched (target) sequence
pct_identity	Percentage of the query sequence that identically matches with the database sequence.
alignment_length	Length of the alignment.
mismatches	Number of base-pair mismatches between the query and the database sequence.
positives	Number of base-pairs with a positive match score between the query and the database sequence.
gap_openings	Number of gaps opened in gapped alignment.
gap_list	List of offsets where a gap is opened.
q_seq_start, q_seq_end	The indices of the portion of the query sequence that is aligned.
q_frame	Translation frame number of the query.
t_seq_start, t_seq_end	The indices of the portion of the target sequence that is aligned.

Table 6–14 Output Parameters for TBLAST_ALIGN Table Function

Parameter	Description
`t_frame`	Translation frame number of the target sequence.
`score`	Score corresponding to the alignment.
`expect`	Expect value of the alignment.

7
Text Mining

The PL/SQL interface enables you to perform Text Mining using a simple two-step process:

Step 1: Given a Text document table, and an Oracle Text Index built against the documents, extract the text "features" using a simple PL/SQL driver provided with the ODM installation (see Section 5.2, "Summary of ODM PL/SQL Sample Programs"). This driver demonstrates how to store all the text features corresponding to a DocID into a table with nested table columns.

Step 2: Provided the table created in Step 1 as input to the `CREATE_MODEL` or `APPLY` operation, as appropriate, to be classified using any classification algorithm — such as SVM, or a clustering algorithm such as *k*-Means, or a feature extraction algorithm such as NMF.

Note that this two-step process is flexible and can handle any general text input; you just have to provide the text features in an input table whose schema corresponds to the one depicted in `dm/demo/sample/plsql/textfe.sql`.

See also Chapter 13, "Text Mining Using ODM," in *Oracle Data Mining Concepts*.

A

Binning

This appendix provides a detailed example of binning.

Table A–1 displays original data before binning. Table A–2 shows the bin boundaries for numeric data; Table A–3 shows bin boundaries for categorical data. Table A–4 shows the results of binning.

Table A–1 Binning Illustration: Data before Binning

PERSON_ID	AGE	WORK CLASS	EDUCATION	MARITAL_ STATUS	OCCUPATION
2	27	Private	HS-grad	Married	Crafts
8	46	Private	Bach.	Separ.	Prof.
10	34	Private	HS-grad	Separ.	Agricultural
11	23	Sta-gov	< Bach.	NeverM	Cleric.
41	30	Private	< Bach.	Married	Sales

Table A–2 Binning Illustration: Bin Boundaries for Numeric Data

COLUMN_ NAME	LOWER_ BOUNDARY	UPPER_ BOUNDARY	BIN_ID	DISPLAY_NAME
AGE	17	24.3	1	17-24.3
AGE	24.3	31.6	2	24.3-31.6
AGE	31.6	38.9	3	31.6-38.9

COLUMN_ NAME	LOWER_ BOUNDARY	UPPER_ BOUNDARY	BIN_ID	DISPLAY_NAME
AGE	38.9	46.2	4	38.9-46.2
AGE	46.2	53.5	5	46.2-53.5

Table A–3 Binning Illustration: Bin Boundaries for Categorical Data

COLUMN_NAME	CATEGORY	BIN_ID	DISPLAY_ NAME
WORKCLASS	Loc-gov	1	Government
WORKCLASS	Fed-gov	1	Government
WORKCLASS	Sta-gov	1	Government
WORKCLASS	Private	2	Others
EDUCATION	HS-grad	1	HS-grad
EDUCATION	< Bach.	2	< Bach.
EDUCATION	Bach.	3	Bach.
EDUCATION	Masters	4	Masters
MARITAL_STATUS	Married	1	Married
MARITAL_STATUS	NeverM	2	NeverM
MARITAL_STATUS	Divorc.	3	Divorc.
MARITAL_STATUS	Widowed	4	Widowed
MARITAL_STATUS	Separ.	5	Separ.
OCCUPATION	Prof	1	Prof
OCCUPATION	Crafts	2	Crafts
OCCUPATION	Exec.	3	Exec.
OCCUPATION	Sales	4	Sales
OCCUPATION	Cleric	5	Cleric
OCCUPATION		6	Other_occ

Table A–4 Binning Illustration: Assignment of Original Data to Bins

PERSON_ID	AGE	WORK CLASS	WEIGHT	EDUCATION	MARITAL_ STATUS	OCCUPATION
2	2	2	2	1	1	2
8	4	2	1	3	5	1
10	3	2	1	1	5	6
11	1	1	1	2	2	5
41	2	2	2	2	1	4

A.1 Use of Automated Binning

The Java interface supports automated binning. An important advantage of automated binning is that it allows ODM to handle raw data. Automated binning also allows initial exploration of problems about which there is little or no information to guide binning decisions.

Currently automatic binning requires closed intervals for numerical bins. This can result in certain values being ignored. For example, if the salary range in the build data table is 0 to 1,000,000, any salary greater than 1,000,000 is ignored when the model is applied. If you are trying to identify likely purchasers of a high-end consumer product, attributes indicating the wealthiest individuals are likely to be deleted, and you probably won't find the best targets. Manual binning has the option of making extreme bins open-ended, that is, with infinite boundaries.

B

ODM Tips and Techniques

This section contains information about some special considerations for clustering models, for SVM models, and for NMF models.

B.1 Clustering Models

ODM supports two algorithms for clustering:

- *k*-Means
- O-Cluster

The two algorithms treat data differently. This section discusses important considerations about data for clustering.

B.1.1 Attributes for Clustering

Binary attributes should have data mining type as follows:

- Numeric for *k*-Means
- Categorical for O-Cluster

B.1.2 Binning Data for *k*-Means Models

You can either bin the data manually or let the algorithm do the binning. For *k*-Means, it is usually best to let the algorithm do the binning. If you bin manually, the first bin number must be 1. We recommend that you have the same number of bins per attribute in order to have the same scale in the distance computation. For example, if age is binned in 20 bins (1...20), and there is a binary attribute (gender), the binary attribute should be binned as 1 and 20 instead of 1 and 2. If this is not done, the algorithm would still work but the results will be unreliable.

B.1.3 Binning Data for O-Cluster Models

You can either bin the data manually or let the algorithm do the binning. For O-Cluster, it is usually best to let the algorithm do the binning. If you bin manually, the first bin number must be 1. In the case of O-Cluster, manual binning should not over-smooth or under-smooth the histograms of numeric attributes. The number of bins does not need to be the same across attributes but should be chosen to capture the attribute value distribution accurately.

B.2 SVM Models

This section describes the ways in which you can affect model build quality and performance with SVM models.

B.2.1 Build Quality and Performance

The user can influence both the SVM model quality (accuracy) and performance (build time) through two basic mechanisms: data preparation and model settings.

Poor choice of settings or data preparation can lead to serious performance degradation. Poor settings choice can also lead to inaccurate models. For example, a model can predict only one class. ODM offer- built in mechanisms that estimate appropriate settings for the problem at hand.

SVM estimates three settings: complexity factor, standard deviation for Gaussian kernels, and epsilon for regression models. These three settings are estimated by default.

Default settings are overridden by specifying a value for the setting when creating the algorithm settings object (Java) or by inserting a row in the settings table for that setting (DBMS_DM).

B.2.2 Data Preparation

Default data preparation is overridden by specifying the data as prepared (ODM). In DBMS_DM there is no default data preparation. Data preparation must be specifically invoked.

ODM and DBMS_DM accept two types of predictors: numeric and categorical. In ODM, the logical data specification identifies the data type of each predictor. DBMS_DM identifies all database numeric types as numeric predictors and all database string types as categorical predictors. SVM requires all predictors to be numeric and the range of values restricted to a small interval around 0. Hence

numeric predictors are normalized and categorical predictors are exploded (described below).

B.2.3 Numeric Predictor Handling

Normalization of numeric predictors is typically required for two reasons: (1) so that the relative influence of the various predictors on the model is not distorted by their relative scaling, and (2) to avoid computational overflow/underflow. To the first point, note that an SVM model (ai) parameter applies to an entire training vector, rather than individual predictors within the vector; hence large individual vector values can dominate. In addition (point 2), note that the kernel computation includes sum of dot products of two potentially large values. Such sums can result in overflow, or, as the exponent of a Gaussian, underflow.

Our offerings for normalization include z-score and min-max normalization. Each normalization technique has advantages and drawbacks. Z-score has the advantage of not distorting the shape of the distribution. However, z-score'd data can still have many large (absolute) values (outside of range {-1, 1}. The number and size of large values can differ greatly across attributes, depending upon their relative non-normality. In addition, the z-score'd range differs from the exploded categorical range {0,1}. Min-max normalization avoids the large value issue and has the same range as the categorical data but may suffer compression of its dense range in the presence of large extreme values. The user could potentially address the min-max normalization drawback with some procedure for handling outliers.

B.2.4 Categorical Predictor Handling

Categorical predictors are exploded into arrays of indicator variables. This is transparent to the user in both interfaces. For example, a predictor, VAR, which takes on one of three possible values: A, B or C becomes three predictors that one could think of as VAR_A, VAR_B and VAR_C. Each of these exploded predictors is a binary attribute with values in the set {0,1}. If VAR_A is 1, then VAR = "A". If VAR_A is 0 then VAR is NOT equal to "A". For multi-valued categorical predictors with no implicit order, explosion, as specified, is the most sensible procedure.

For binary predictors or multi-valued categoricals with an implicit order there are choices. Consider a predictor taking on values in the set {LOW, MEDIUM, HIGH}. The user could choose to recode the values to {0, 0.5, 1}, or, use some other mapping that is intended to reflect the relative distance between the categories. The recoded predictor would then be passed as numeric field.

Binary predictors take on one of two possible values. E.g. a binary predictor might take on values from the set {NO, YES}. The decision to recode such predictors to {0, 1} depends upon whether there is a preferred positive value or not.

B.2.5 Regression Target Handling

In the Java API, for performance and accuracy, we internally normalize regression targets. In DBMS_DM, the user can choose to normalize the target externally. Consider a regression target with large absolute values. Because the predictors are constrained, the coefficients (ai) must be large, or the number of non-zero coefficients must be large, or both, otherwise it is impossible to predict a large value. The range of the coefficients is restricted by the ComplexityFactor (C) setting,. With a small C, the performance and, possibly, the accuracy of SVM can be poor. With a large C, the training time can increase significantly. In addition, the model can be larger than would be otherwise necessary. To avoid this issue we normalize the targets. Since the goal is to avoid large values, min-max normalization is often a good choice (see numeric predictor handling discussion above).

B.2.6 SVM Algorithm Settings

Several of the algorithm settings can significantly affect SVM accuracy and performance. The issues and considerations in choosing setting values are discussed briefly in the sections below. In the literature, cross-validation and grid search techniques are often used to select parameters. These methods, while accurate, are very expensive. Rather than saddle users with an expensive procedure, we have opted for computationally inexpensive procedures that fit easily within our implementation framework. The intent is to provide settings that should give the model adequate complexity for the problem at hand. If the user requires greater accuracy and is willing to incur the additional computational expense, our defaults can be used as the starting point for a grid search.

B.2.7 Complexity Factor (C)

The complexity factor, C, determines the trade-off between minimizing model error on the training data and minimizing model complexity. Its responsibility is to avoid over-fit (an over-complex model fitting noise in the training data) and under-fit (a model that is too simple). Overfit exists if the model fits the training data well but does poorly on held-aside test data. Underfit exists if the model does poorly on both training and test data. If the user wishes to override the default C after seeing the model result, the user can obtain the internally computed default value as a starting

point for a grid search. Both the java API and PL/SQL interfaces have methods for getting the Complexity factor. The subsequent discussion provides qualitative description of the impact of C on the model build.

Very large value of C places extreme penalty on errors, so that SVM seeks a perfect separation of target classes, or, in the case of regression, a perfect (epsilon-insensitive — see below) fit. Assuming the data to have at least some noise, this is over-fit. Large C leaves the parameters of the model (ai) unconstrained, i.e., the model is complex with high capacity for fitting data. Conversely, small C places low penalty on errors and high constraints on the model parameters, which can lead to under-fit.

Standard Deviation — Gaussian Kernels Only

Standard deviation is an SVM setting applying to Gaussian Kernels only. This parameter, in conjunction with C, affects the trade-off between error on the training data and generalization. For fixed C, underfit results as the standard deviation gets large, and overfit results as the standard deviation goes to zero.

To facilitate using the default values computed in ODM as a starting point, methods exist to extract the setting values.

B.2.8 Epsilon — Regression Only

Epsilon is an SVM setting applying to regression models only. The parameter separates small errors from large errors. Small errors are considered to have no associated penalty. Only large errors are penalized. In the Java API, if the default target normalization is used the value of epsilon is re-scaled accordingly. In DBMS_DM, epspilon needs to be re-scaled by the user. By default, epsilon is estimated internally.

The epsilon parameter affects the number of support vectors and, thus, indirectly, the trade-off between model complexity and generalization (over-fit and under-fit as possible consequences). An estimate of the standard deviation of the additive noise in the target variable is needed to find an appropriate epsilon value and thus minimize predictive error. The estimate can be based either on domain knowledge or it can be obtained from the residuals of a crude model (e.g., polynomial, KNN) built on the training data.

B.2.9 Kernel Cache — Gaussian Kernels Only

The most expensive operation in building a gaussian SVM model is the computation of kernels. The general approach taken to build is to converge within a

chunk of data at a time, then to test for violators outside of the chunk. Build is complete when there are no more violators within tolerance. The size of the chunk is chosen such that the associated kernels can be maintained in memory in a "Kernel Cache". The larger the chunk size, presumably, the better the chunk represents the population of training data and the fewer number of times new chunks will need to be created. Generally, larger caches imply faster builds. Default size is 50M.

B.2.10 Tolerance

Tolerance is the maximum size of a violation of convergence criteria such that the model is considered to have converged. The default value is 0.001. Larger values imply faster building but less accurate models.

B.3 NMF Models

Traditionally, as part of standard numerical analysis, matrix factorization is a common preprocessing procedure prior to solving a liner system of equations. For data mining, matrix factorization offers a way to reduce the dimensionality of a dataset and extract features that reveal interesting structure in the data or provide inputs to further types of analysis. In matrix factorization, the number of the dataset- independent columns is reduced by projection onto a lower dimensional space (e.g., smaller matrices).

Rank reduction by factorization can reveal interesting low-dimensional subspaces embedded in large dimensionality datasets space and is a useful operation for pattern discovery and feature extraction. For example, the traditional Principal Component Analysis (PCA) uses a projection of the data on dimensions along which it varies the most and can be used to visualize the most dominant structure in a dataset.

Non-negative matrix factorization (NMF), by imposing non-negativity constraints on the factors, has been shown to be a useful decomposition and feature extraction method in fields such as object detection and recognition and to be a valuable alternative PCA[1]. By forcing a dataset (matrix) to "fit" into a product of smaller datasets (matrices), NMF compresses the data and tends to eliminate some of the redundancies and expose the most common patterns. By using a parts-based or component-based decomposition, and in contrast to PCA and other techniques, the compressed version of the data is not random looking and can be used to understand interesting patterns and common trends in the dataset. The NMF

[1] Daniel D. Lee, and H. Sebastian Seung, Learning the parts of objects by non-negative matrix factorization. *Nature* 1999, Oct 21, 401(6755), 788-793.

decomposition also induces a numerical taxonomy that can be used for grouping the rows or columns of the original dataset. The extracted features can be used as inputs to other analysis tasks such as classification or indexing. This procedure has proven useful in face recognition problems or the discovery of semantic features in texts[2].

Given an N (rows) x M (columns) 2D dataset A and $k < N, M$, NMF computes an approximation of the original data, $A \sim W H$, where W is N by k, and H is k by M. Starting from random initial conditions, W and H are iteratively updated until convergence to a local minimum is achieved, monitored by the minimization of the Euclidean cost function. A must have positive entries, and so are W and H by construction. Even though localization is not an explicit property of the algorithm, NMF appears to produce quite localized and sparse features that facilitate the interpretation of results and the transparency of the model. For example, when NMF is applied to a dataset of facial images, the extracted features are facial parts: eyes, noses, etc. When the dataset is a document/keyword matrix, then NMF extracts "semantic" features[3].

When NMF is used as a feature extractor, it can benefit from scaling individual attributes to a common scale via normalization. This would facilitate pattern discovery and make dimensionality reduction more effective. A preferred approach would be to perform outlier treatment (e.g., by using a winsorizing transformation) and then perform min-max normalization. Having individual attributes on a common scale would help ranking and interpretation of feature coefficients in terms of there relative magnitude. Additionally, applying a normalization transformation would allow NMF to operate on negative data as well. Otherwise, the data need to be shifted to the positive range manually on a per-attribute basis.

If NMF is used to cluster column instances -— e.g., by using the amplitudes from the rows of H— then according to the nature of the problem, one may consider normalizing the rows, the columns, or both, prior to using NMF.

[2] Same as footnote 1, above.

[3] Same as footnote 1, above.

Index

A

B

C

D

P

R

S

T

U

W