Oracle9*i* Data Mining

Concepts

Release 9.2.0.2

October 2002

Part No. A95961-02

ORACLE

Oracle9*i* Data Mining Concepts, Release 9.2.0.2

Part No. A95961-02

Contents

Send Us Your Comments

Oracle9*i* Data Mining Concepts, Release 9.2.0.2

Part No. A95961-02

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:

- FAX: 781-238-9893 Attn: Oracle9*i* Data Mining Documentation
- Postal service:
  Oracle Corporation
  Oracle9*i* 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 is a revised edition of *Oracle9i Data Mining Concepts*, originally published in March 2002.

This manual describes how to use the Oracle9i Data Mining Java Application Programming Interface to perform data mining tasks, including building and testing models, computing lift, and scoring.

Intended Audience

This manual is intended for anyone planning to write Java programs using the Oracle9*i* Data Mining API. Familiarity with Java, databases, and data mining is assumed.

Structure

This manual is organized as follows:

- Chapter 1: Defines basic data mining concepts.
- Chapter 2: Describes compiling and executing ODM programs and using ODM to perform common data mining tasks.
- Chapter 3: Contains short examples of using ODM to build a model and then using that model to score new data.
- Appendix A: Lists ODM sample programs and outlines how to compile and execute them.
- Glossary: A glossary of terms related to data mining and ODM.

Where to Find More Information

The documentation set for Oracle9*i* Data Mining is part of the *Oracle9i Database Documentation Library*. The ODM documentation set consists of the following documents, available online:

- *Oracle9i Data Mining Administrator's Guide*, Release 2 (9.2)
- *Oracle9i Data Mining Concepts*, Release 9.2.0.2 (this document)

For last minute information about ODM, see the *Oracle9i README*, Release 9.2.0.2, and the release notes for your platform.

For detailed information about the ODM API, see the ODM Javadoc in the directory `$ORACLE_HOME/dm/doc` on any system where ODM is installed.

Related Manuals

For more information about the database underlying Oracle9*i* Data Mining, see:

- *Oracle9i Administrator's Guide*, Release 2 (9.2)

For information about upgrading from Oracle9*i* Data Mining release 9.0.1 to release 9.2.0, see

- *Oracle9i Database Migration*, Release 2 (9.2)

For information about installing Oracle9*i* Data Mining, see

- *Oracle9i Installation Guide*, Release 2 (9.2)

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 Oracle9*i* is referred to as SQL. This interface is the Oracle9*i* 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.
< >	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.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

1

Basic ODM Concepts

Oracle9*i* Data Mining (ODM) embeds data mining within the Oracle9*i* database. The data never leaves the database — the data, data preparation, model building, and model scoring activities all remain in the database. This enables Oracle9*i* to provide an infrastructure for data analysts and application developers to integrate data mining seamlessly with database applications.

Data mining functions such as model building, testing, and scoring are provided via a Java API. This chapter provides an overview of basic Oracle9*i* Data Mining concepts. It is organized as follows:

1.1 New Features and Functionality

With Release 2, Oracle9*i* Data Mining adds several data mining capabilities: Adaptive Bayes Network, clustering, attribute importance (also known as feature selection), and others, as described below.

- **Adaptive Bayes Networks (ABN):** Expands ODM support of supervised learning techniques (techniques that predict a target value). ODM can be used to make predictions with an associated probability.

  A significant benefit of ABN is that they produce a set of human-readable "rules" or explanations that can be interpreted by analysts and managers. Users can then query the database for all records that fit the criteria of a rule.

- **Clustering:** Expands ODM support of unsupervised learning (learning techniques that do not have a target value). Clustering can be used to segment data into naturally occurring clusters or for assigning new data to clusters. ODM Clustering techniques use *k*-means and an Oracle proprietary algorithm, O-Cluster, that allows both numerical and categorical data types to be clustered. The clustering model generates probabilistic cluster membership assignment and cluster rules that describe the characteristics of each cluster.

- **Attribute Importance:** Used to identify those attributes that have the greatest influence on a target attribute. It assesses the predictive usefulness of each available non-target mining attribute and ranks them according to their predictive importance. See Section 1.3.4, "Attribute Importance". Attribute importance is also sometimes referred to as *feature selection* or *key fields*.

- **Model Seeker:** A productivity tool that automatically builds multiple data mining models with minimal user input, compares the models, and selects the "best" of the models it has built. See Section 1.4.3, "Model Seeker", for a fuller description.

- **Automated Binning:** Automates the task of discretizing (binning) all attributes into categorical bins for the purposes of counting. Internally, many ODM algorithms require the data to be binned for analysis. With this feature, the user can create bins of fixed size for each field. The user can either bin the data as part of data preprocessing or allow the algorithms to bin the data automatically. With manual preprocessing, the user sets bin boundaries and can later modify them. With automatic preprocessing, there is no modifying the boundaries after they are set. Target attribute values are not binned. See Section 1.8, "Discretization (Binning)".

- **Predictive Model Markup Language (PMML):** ODM supports the import and export of PMML models for Naive Bayes and Association Rules models. PMML

allows data mining applications to produce and consume models for use by data mining applications that follow the PMML 2.0 standard. See Section 1.9, "PMML Support".

- **Mining Task:** All data mining operations (build, test, compute lift, apply, import, and export) are performed asynchronously using a mining task. This is important when you are creating large data mining applications. The static methods supported in ODM release 9.0.1 for these mining operations are not supported in this release. Mining tasks allow the user to obtain the status of the mining operations as they are executed.

1.2 Oracle9*i* Data Mining Components

Oracle9*i* Data Mining has two main components:

- Oracle9*i* Data Mining API
- Data Mining Server (DMS)

1.2.1 Oracle9*i* Data Mining API

The Oracle9*i* Data Mining API is the component of Oracle9*i* Data Mining that allows users to write Java programs that mine data.

The ODM API provides an early look at concepts and approaches being proposed for the emerging standard Java Data Mining (JDM). JDM follows Sun Microsystem's Java Community Process as a Java Specification Request (JSR-73). JDM used design elements from several evolving data mining standards, including the Object Management Group's Common Warehouse Metadata (CWM), the Data Mining Group's Predictive Model Markup Language (PMML), and the International Standards Organization's SQL/MM for Data Mining. JDM has also influenced these standards. Oracle9*i* Data Mining will comply with the JDM standard when that standard is published.

1.2.2 Data Mining Server

The Data Mining Server (DMS) is the server-side, in-database component that performs the data mining operations within the 9*i* database, and thus benefits from RDBMS availability and scalability.

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.

1.3 Data Mining Functions

Data mining models are based on one of two kinds of learning: *supervised* and *unsupervised* (sometimes referred to as directed and undirected learning). Supervised learning functions are typically used to predict a value.Unsupervised learning functions are typically used to find the intrinsic structure, relations, or affinities in a body of data but no classes or labels are assigned a priori. Examples of unsupervised learning algorithms include *k*-means clustering and Apriori association rules. An example of supervised learning algorithms includes Naive Bayes for classification.

ODM supports the following data mining functions:

- Classification (supervised)
- Clustering (unsupervised)
- Association Rules (unsupervised)
- Attribute Importance (supervised)

1.3.1 Classification

In a classification problem, you have a number of cases (examples) and wish to predict which of several classes each case belongs to. Each case consists of multiple attributes, each of which takes on one of several possible values. The attributes consist of multiple predictor attributes (independent variables) and one target attribute (dependent variable). Each of the target attribute's possible values is a class to be predicted on the basis of that case's predictor attribute values.

1.3.1.1 Costs

Classification is used in customer segmentation, business modeling, credit analysis, and many other applications. For example, a credit card company may wish to predict which customers will default on their payments. Each customer corresponds to a case; data for each case might consist of a number of attributes that describe the customer's spending habits, income, demographic attributes, etc. These are the predictor attributes. The target attribute indicates whether or not the customer has defaulted; that is, there are two possible classes, corresponding to having defaulted or not. The build data are used to build a model that you then use to predict, for new cases, whether those customers are likely to default.

A classification task begins with build data for which the target values (or class assignments) are known. Different classification algorithms use different techniques for finding relations between the predictor attributes' values and the target attribute's values in the build data. These relations are summarized in a model, which can then be applied to new cases with unknown target values to predict target values. A classification model can also be used on build data with known target values, to compare the predictions to the known answers. This technique is used when testing a model to measure the model's predictive accuracy. The application of a classification model to new data is often called *scoring the data*.

In a classification problem, it may be important to specify the costs involved in making an incorrect decision. Doing so can be useful when the costs of different misclassifications varies significantly.

For example, suppose the problem is to predict whether a user will respond to a promotional mailing. The target has two categories: YES (the customer responds) and NO (the customer does not respond). Suppose a positive response to the promotion generates $500 and that it costs $5 to do the mailing. If the model predicts YES and the actual value is YES, the cost of misclassification is $0. If the model predicts YES and the actual value is NO, the cost of misclassification is $5. If the model predicts NO and the actual value is YES, the cost of misclassification is $500. If the model predicts NO and the actual value is NO, the cost is $0.

The row indexes of a cost matrix correspond to *actual values*; the column indexes correspond to *predicted values*. For any pair of actual / predicted indexes, the value indicates the number of records classified in that pairing.

Some algorithms, like Adaptive Bayes Network, optimize for the cost matrix directly, modifying the model structure so as to produce minimal cost solutions. Other algorithms, like Naive Bayes, that predict probabilities, use the cost matrix during scoring to propose the least expensive solution.

1.3.1.2 Priors

In building a classification model, you may need to balance the number of positive and negative cases for the target of a supervised model. This can happen either because a given target value is rare in the population, for example, fraud cases, or because the data you have does not accurately reflect the real population, that is, the data sample is skewed.

A classification model works best when it has a reasonable number of examples of each target value in its build data table. When only a few possible values exist, it works best with more or less equal numbers of each value.

For example, a data table may accurately reflect reality, yet have 99% negatives in its target classification and only 1% positives. A model could be 99% accurate if it predicted on the negative case, yet the model would be useless.

To work around this problem, you can create a build data table in which positive and negative target values are more or less evenly balanced, and then supply priors information to tell the model what the true balance of target values is.

1.3.2 Clustering

Clustering is a technique useful for exploring data. It is particularly useful where there are many cases and no obvious natural groupings. Here, clustering data mining algorithms can be used to find whatever natural groupings may exist.

Clustering analysis identifies clusters embedded in the data. A cluster is a collection of data objects that are similar in some sense to one another. A good clustering method produces high-quality clusters to ensure that the inter-cluster similarity is low and the intra-cluster similarity is high; in other words, members of a cluster are more like each other than they are like members of a different cluster.

Clustering can also serve as a useful data-preprocessing step to identify homogeneous groups on which to build predictive models. Clustering models are different from predictive models in that the outcome of the process is not guided by a known result, that is, there is no target attribute. Predictive models predict values for a target attribute, and an error rate between the target and predicted values can be calculated to guide model building. Clustering models, on the other hand, uncover natural groupings (clusters) in the data. The model can then be used to assign groupings labels (cluster IDs) to data points.

In ODM a cluster is characterized by its *centroid*, attribute histograms, and place in the clustering model hierarchical tree. ODM performs hierarchical clustering using an enhanced version of the *k*-means algorithm and O-Cluster, an Oracle proprietary algorithm. The clusters discovered by these algorithms are then used to create rules that capture the main characteristics of the data assigned to each cluster. The rules represent the hyperboxes (bounding boxes) that envelop the clusters discovered by the clustering algorithm. The antecedent of each rule describes the clustering bounding box. The consequent encodes the cluster ID for the cluster described by the rule. For example, for a dataset with two attributes: AGE and HEIGHT, the following rule represents most of the data assigned to cluster 10:

```
If AGE >= 25 and AGE <= 40
and HEIGHT >= 5.0ft
and HEIGHT <= 5.5ft
then CLUSTER = 10
```

The clusters are also used to generate a Bayesian probability model which is used during scoring for assigning data points to clusters.

1.3.3 Association Rules

The Association Rules model is often associated with "market basket analysis", which is used to discover relationships or correlations among a set of items. It is widely used in data analysis for direct marketing, catalog design, and other business decision-making processes. A typical association rule of this kind asserts the likelihood that, for example,"70% of the people who buy spaghetti, wine, and sauce also buy garlic bread."

Association rules capture the co-occurrence of items or events in large volumes of customer transaction data. Because of progress in bar-code technology, it is now possible for retail organizations to collect and store massive amounts of sales data, referred to as "basket data." Association rules were initially defined on basket data, even though they are applicable in several other applications. Finding all such rules is valuable for cross-marketing and mail-order promotions, but there are other applications as well: catalog design, add-on sales, store layout, customer segmentation, web page personalization, and target marketing.

Traditionally, association rules are used to discover business trends by analyzing customer transactions. However, they can also be used effectively to predict Web page accesses for personalization. For example, assume that after mining the Web access log we discovered an association rule "A and B implies C," with 80% confidence, where A, B, and C are Web page accesses. If a user has visited pages A and B, there is an 80% chance that he/she will visit page C in the same session. Page C may or may not have a direct link from A or B. This information can be used to create a link dynamically to page C from pages A or B so that the user can "click-through" to page C directly. This kind of information is particularly valuable for a Web server supporting an e-commerce site to link the different product pages dynamically, based on the customer interaction.

Association rule mining can be formally defined as follows: Let $I = \{i_1, i_2, ..., i_n\}$ be a set of literals (constants: either a number or a character) called *items* and D be a set of *transactions* where each transaction T is a set of items such that T is a subset of I. Associated with each transaction is an identifier, called its TID. An association rule is an implication of the form X implies Y, where X and Y are both subsets of I, and X intersect Y is empty. The rule has support *s* in the database D if *s*% of the transactions in D contain both X and Y, and confidence *c* if *c*% of transactions that contain X also contain Y. The problem of mining association rules is to generate all rules that have support and confidence greater than the user-specified minimum support and minimum confidence, respectively.

Algorithms that calculate association rules work in two phases. In the first phase, all combinations of items that have the required minimum support (called the "frequent item sets") are discovered. In the second phase, rules of the form X implies Y with the specified minimum confidence are generated from the frequent item sets. Typically the first phase is computationally expensive and has in recent years attracted attention from researchers all over the world. This has resulted in several innovative techniques for discovering frequent item sets.

There are several properties of association rules that can be calculated. ODM supports two:

- **Support:** Support of a rule is a measure of how frequently the items involved in it occur together. Using probability notation, support (A implies B) = P(A, B).
- **Confidence:** Confidence of a rule is the conditional probability of B given A; confidence (A implies B) = P (B given A), which is equal to P(A, B) or P(A).

These statistical measures can be used to rank the rules and hence the predictions.

1.3.4 Attribute Importance

Attribute Importance, also known as *feature selection*, provides an automated solution for improving the speed and possibly the accuracy of classification models built on data tables with a large number of attributes.

Attribute Importance ranks the predictive attributes by eliminating redundant, irrelevant, or uninformative attributes and identifying those predictor attributes that may have the most influence in making predictions. ODM examines data and constructs classification models that can be used to make predictions about subsequent data. The time required to build these models increases with the number of predictors. Attribute Importance helps a user identify a proper subset of these attributes that are most relevant to predicting the target. Model building can proceed using the selected attributes (predictor attributes) only.

Using fewer attributes decreases model building time, although sometimes at a cost in predictive accuracy. Using too many attributes (especially those that are "noise") can affect the model and degrade its performance and accuracy. By extracting as much information as possible from a given data table using the smallest number of attributes, a user can save significant computing time and often build better models.

Attribute Importance permits the user to specify a number or percentage of attributes to use; alternatively the user can specify a cutoff point. After an Attribute Importance model is built, the user can select the subset of attributes based on the ranking or the predictive value.

Attribute Importance can be applied to data tables with a very large set of attributes. However, the DBA may have to tune the database in various ways to ensure that a large Attribute Importance build executes efficiently. For example, it is important to ensure that there is adequate swap space and table space.

1.4 ODM Algorithms

Oracle9i Data Mining supports the following data mining algorithms:

- Adaptive Bayes Network (classification)
- Naive Bayes (classification)
- Model Seeker (classification)
- *k*-Means (clustering)
- O-Cluster (clustering)
- Predictor variance (attribute importance)
- Apriori (association rules)

The choice of data mining algorithm depends on the data and the conclusions to be reached.

For classification:

- Choose ABN if you
  - have a large number of attributes
  - need model *transparency*, that is, rules that explain the model
  - want more options to control the amount of time required to build the model
- Choose NB for the fastest build time
- Choose Model Seeker if you
  - are unsure which settings should be provided
  - wish to compare Naive Bayes to Adaptive Bayes Network automatically
  - the *figure of merit* for computing the "best" model is appropriate for your situation

For clustering:

- Choose O-Cluster if you
  - want the number of clusters to be automatically determined
  - have both categorical and numerical attributes

  - have a large number of attributes >20)
  - have a large number of cases (>1000)
- Choose *k*-means if you
  - want to specify the number of clusters
  - need to mine only numerical attributes
  - have small tables (<100 rows)
  - a small number of attributes (<100)

1.4.1 Adaptive Bayes Network

Adaptive Bayes Network (ABN) is an Oracle proprietary algorithm supporting decision-tree-like features in that it produces "rules". ABN provides a fast, scalable, non-parametric means of extracting predictive information from data with respect to a target attribute. (Non-parametric statistical techniques avoid assuming that the population is characterized by a family of simple distributional models, such as standard linear regression, where different members of the family are differentiated by a small set of parameters.)

ABN can provide such information in the form of human-understandable rules. For example, a rule may be "If income is $70K-$80K and household size is 3-5, the likelihood of owning a late-model minivan is YES." The rules produced by ABN are one of its main advantages over Naive Bayes. The business user, marketing professional, or business analyst can understand the basis of the model's predictions and can therefore be comfortable acting on them and explaining them to others.

In addition to explanatory rules, ABN provides performance and scalability, which are derived via a collection of user parameters controlling the trade-off of accuracy and build time.

ABN predicts binary as well as multiclass targets. Binary targets are those that take on only two values, for example, *buy* and *not buy*. Multiclass targets have more than two values, for example, products purchased (product A or product B or product C). Multiclass target values are not assumed to exist in an ordered relation to each other, for example, hair brush is not assumed to be greater or less than comb.

ABN can use costs and priors for both building and scoring (see Section 1.3.1.1, "Costs" and Section 1.3.1.2, "Priors").

A key concept for ABN is *network feature*. Network features are like individual decision trees. Features are tree-like multi-attribute structures. From the standpoint of the network, features are conditionally independent components. Features

contain at least one attribute (the root attribute). Conditional probabilities are computed for each value of the root predictor. A two-attribute feature will have, in addition to the root predictor conditional probabilities, computed conditional probabilities for each combination of values of the root and the depth 2 predictor. That is, if a root predictor, *x*, has *i* values and the depth 2 predictor, *y*, has *j* values, a conditional probability is computed for each combination of values {*x=a, y=b* such that *a* is in the set [1,..,*i*] and *b* is in the set [1,..,*j*]}. Similarly, a depth 3 predictor, *z*, would have an additional associated conditional probability computed for each combination of values {*x=a, y=b, z=c* such that *a* is in the set [1,..,*i*] and *b* is in the set [1,..,*j*] and *c* is in the set [1,..,*k*]}.

1.4.1.1 Build Parameters

To control the execution time of a build, ABN provides four user-settable parameters:

- **MaximumNetworkFeatureDepth**: This parameter restricts the depth of any individual network feature in the model. At each depth for an individual network feature, there is only one predictor. Each depth level requires a scan of the data to accumulate the counts required for predictor selection and probability estimates and an apply operation on a sample to test for significance. Thus, the computational cost of deep feature builds may be high. The range for this parameter consists of the positive integers. The NULL or 0 value setting has special meaning: unrestricted depth. Builds beyond depth 7 are rare. Setting this parameter to 1 makes the algorithm act like a Naive Bayes model with stepwise attribute selection. ABN may stop model building well before reaching the maximum. The default is 10.
- **MaximumNumberOfNetworkFeatures:** This controls the maximum number of features included in the model. It also controls the number of predictors in the Naive Bayes model it tests as a first step in its model selection procedure. Subsequent steps in the model build procedure construct multidimensional features by extending single-predictor "seed" features. Note that the seed features are extended in rank order. During stepwise selection, subsequent features must improve the model as measured by MDL (Minimum Description Length) relative to the current state of the model. Thus the likelihood of substantial benefit from extending later features declines rapidly. The default is 10.
- **MaximumConsecutivePrunedNetworkFeatures:** This is the maximum number of consecutive pruned features before halting the stepwise selection process. A negative value of –1 is used to indicate that only the Naive Bayes model and a single-feature model are constructed. If the Naive Bayes model is best, then it is

selected. Otherwise, all as-yet untested features are pruned from the final feature tree array. The default is –1.

- **MaximumBuildTime:** The maximum build time (in minutes) allows the user to build quick, possibly less accurate models for immediate use or simply to get a sense of how long it will take to build a model with a given set of data. To accomplish this, the algorithm divides the build into milestones (model states) representing complete functional models. The algorithm completes at least a single milestone and then projects whether it can reach the next one within the user-specified maximum build time. This decision is revisited at each milestone achieved until either the model build is complete or the algorithm determines it cannot reach the next milestone within the user-specified time limit. The user has access to the statistics produced by the time estimation procedure. The default is NULL (no time limit):

  Model States:

  – **CompleteMultiFeature:** Multiple features have been tested for inclusion in the model. MDL pruning has determined whether the model actually has one or more features. The model may have completed either because there is insufficient time to test an additional feature or because the number of consecutive features failing the stepwise selection criteria exceeded the maximum allowed or seed features have been extended and tested.

  – **CompleteSingleFeature:** A single feature has been built to completion.

  – **IncompleteSingleFeature:** The model consists of a single feature of at least depth two (two predictors) but the attempts to extend this feature have not completed.

  – **NaiveBayes:** The model consists of a subset of (single-predictor) features that individually pass MDL correlation criteria. No MDL pruning has occurred with respect to the joint model.

  The algorithm outputs its current model state and statistics that provide an estimate of how long it would take for the model to build (and prune) a feature.

  See Table 1–1, below, for a comparison of the main characteristics of the two classification algorithms, Adaptive Bayes Network and Naive Bayes.

1.4.2 Naive Bayes Algorithm

The Naive Bayes algorithm (NB) makes predictions using Bayes' Theorem, which derives the probability of a prediction from the underlying evidence, as described below. NB affords fast model building and scoring.

NB can be used for both binary and multiclass classification problems to answer questions such as "Which customers will switch to a competitor? Which transaction patterns suggest fraud? Which prospects will respond to an advertising campaign?" For example, suppose a bank wants to promote its mortgage offering to its current customers and that, to reduce promotion costs, it wants to target the most likely prospects. The bank has historical data for its customers, including income, number of household members, money-market holdings, and information on whether a customer has recently obtained a mortgage through the bank. Using NB, the bank can predict how likely a customer is to respond positively to a mortgage offering. With this information, the bank can reduce its promotion costs by restricting the promotion to the most likely candidates.

Bayes' Theorem proves the following equation:

$$P(\text{this-prediction} \mid \text{this-evidence}) = \frac{P(\text{this-prediction})\, P(\text{this-evidence} \mid \text{this-prediction})}{\text{sum}P(\text{some-prediction})\, P(\text{this-evidence} \mid \text{some-prediction})}$$

where P means "probability of", " | " means "given", and "sum" means "sum of all these terms". Translated into English, the equation says that the probability of a particular predicted event, given the evidence in this instance, is computed from three other numbers: the probability of that prediction in similar situations in general, ignoring the specific evidence (this is called the prior probability); times the probability of seeing the evidence we have here, given that the particular prediction is correct; divided by the sum, for each possible prediction (including the present one), of a similar product for that prediction (i.e., the probability of that prediction in general, times the probability of seeing the current evidence given that possible prediction).

NB assumes that each attribute, or piece of evidence, is independent from the others. In practice, this assumption usually does not degrade the model's predictive accuracy significantly, and makes the difference between a computationally feasible algorithm and an intractable one.

It is useful to have a good estimate of the accuracy of any predictive model. An especially accurate estimate of accuracy is a type of cross-validation called "leave-one-out cross-validation", discussed below.

Naive Bayes cross-validation permits the user to test model accuracy on the same data that was used to build the model, rather than building the model on one portion of the data and testing it on a different portion. Not having to hold aside a portion of the data for testing is especially useful if the amount of build data is relatively small.

"Leave-one-out cross-validation" is a special case of cross-validation in which one record is left out of the build data when building each of several models. The

number of models built equals the number of records (omitting a different build record for each model), which makes this procedure computationally expensive. With Naive Bayes models, however, the approach can be modified such that all build records are used for building a single model. Then, the model is repeatedly modified to quickly remove the effects of one build record, incrementally "unbuilding" the model for that record, as though that record had been omitted when building the model in the first place. The accuracy of the prediction for each build record can then be assessed against the model that would have been built from all the build records except that one, without having had to actually build a separate model for each build record.

To use Naive Bayes cross-validation, the user executes a MiningTaskCrossValidate task, specifying that a Naive Bayes model is to be built and tested. The execution of the cross-validate task creates a MiningTestResult object populated with the test results.

See Table 1–1, below, for a comparison of the main characteristics of ABN and NB.

Table 1–1 Comparison of Adaptive Bayes Network and Naive Bayes

Feature	Adaptive Bayes Network	Naive Bayes
Number of cases	Any size	Any size
Number of attributes	Any number (built-in feature selection)	Best if less than 200
Speed	Not as fast	Faster
Accuracy	As accurate or more accurate than Naive Bayes	As accurate or less accurate than Adaptive Bayes Network
Attribute types	Numerical (binned) and categorical	Numerical (binned) and categorical
Automatic binning	Yes	Yes
Target attribute	Binary and multiclass	Binary and multiclass

1.4.3 Model Seeker

Model Seeker is a new feature of the ODM API that allows a user to build multiple classifications, evaluate the models, and select a "best" model, asynchronously.

The models to be built and evaluated can be a combination of Naive Bayes (NB) and Adaptive Bayes Network (ABN) models. Model Seeker does not build unsupervised models.

After building the specified models, Model Seeker evaluates each model by testing and calculating lift. Model Seeker generates a summary of information about each model built so that a user can manually select the "best" model using different criteria, if desired.

Model Seeker's criterion for the "best" model is the one with the largest value for the weighted target positive and total negative relative error rate. The weight is set as the relative importance of the positive category to the other categories treated as a single negative category. If the weight is set to 1.0, the positive category error rate has the same weight as all the other categories combined.

The following formula is used to calculate the figure of merit (FOM) for the "best" model, where FOM is the weighted sum of target positive relative accuracy and total negative relative accuracy:

$$FOM = \frac{W * (\text{number of correct positives})}{(W + 1) * (\text{number of actual positives})} + \frac{(\text{number of correct negatives})}{(W + 1) * (\text{number of actual negatives})}$$

where W is the user-specified *weight*, a value that must be ≥ 0. The weight is the ratio of the false negative cost to the false positive cost. A weight of 1 means that the false positives and false negatives have equal weight.

1.4.4 Enhanced *k*-Means Algorithm

The *k*-means algorithm is a distance-based clustering algorithm that partitions the data into a predetermined number of clusters (provided there are enough distinct cases). The *k*-means algorithm works only with numerical attributes. Distance-based algorithms rely on a distance metric (function) to measure the similarity between data points. Data points are assigned to the nearest cluster according to the distance metric used.

ODM implements a hierarchical version of the *k*-means algorithm. The tree can either be grown one level at a time (balanced approach) or one node at the time (unbalanced approach). The node with the largest distortion (sum of distance to the node's centroid) is split to increase the size of the tree until the desired number of clusters is reached.

This incremental approach to *k*-means avoids the need for building multiple *k*-means models and provides clustering results that are consistently superior to the traditional *k*-means.

The choice between balanced and unbalanced approaches is controlled by the system parameter CL_ALG_SETTING_TREE_GROWTH in the ODM_CONFIGURATION table. The balanced approach is faster than the

unbalanced approach, while the unbalanced approach generates models with smaller overall distortion.

1.4.4.1 Binning for *k*-Means

ODM-enhanced *k*-means bins the data internally, thus providing automatic data discretization. However, if manual binning is used, the bin values should be represented by contiguous integer numbers starting at 1. In addition, the same number of bins should be used for all attributes.

k-means works with numerical data only. As a result, if a user would like to cluster data with categorical attributes, he/she should "explode" the categorical attribute into multiple binary columns (one per unique value of the categorical attribute) before using ODM clustering *k*-means.

1.4.4.2 Scalability through Summarization

Because traditional *k*-means requires multiple passes through the data, it can be impractical for large data tables that don't fit in memory. In this case multiple expensive database scans would be required. ODM's enhanced *k*-means requires at most one database scan. For data tables that don't fit in memory, the enhanced *k*-means algorithm employs a smart summarization approach that creates a summary of the data table that can be stored in memory. This approach allows the enhanced *k*-means algorithm to handle data tables of any size. The summarization scheme can be seen as a smart sampling approach that first identifies the main partitions in the data and then generates summary points for each partition in proportion to their share of the total data. Each summary point has a weight that accounts for the proportion of the data it represents.

1.4.4.3 Scoring

The clusters discovered by enhanced *k*-means are used to generate a Bayesian probability model that is then used during scoring (model apply) for assigning data points to clusters. The traditional *k*-means algorithm can be interpreted as a mixture model where the mixture components are spherical multivariate normal distributions with the same variance for all components. (A mixture model is a type of density model that includes several component functions (usually Gaussian) that are combined to provide a multimodal density.)

In the mixture model created from the clusters discovered by enhanced *k*-means, on the other hand, the mixture components are a product of independent normal distribution with potentially different variances. Because of this greater flexibility, the probability model created by enhanced *k*-means provides a better description of the underlying data than the underlying model of traditional *k*-means.

See Table 1–2, below, for a comparison of the main characteristics of the two clustering algorithms.

1.4.5 O-Cluster Algorithm

The O-Cluster algorithm creates a hierarchical grid-based clustering model, that is, it creates axis-parallel partitions in the input attribute space. The algorithm operates recursively. The resulting hierarchical structure represents an irregular grid that tessellates the attribute space into clusters. The resulting clusters define dense areas in the attribute space. The clusters are described by intervals along the attribute axes and the corresponding centroids and histograms. A parameter called *sensitivity* defines a baseline density level. Only areas with peak density above this baseline level can be identified as clusters.

1.4.5.1 Binning for O-Cluster

O-Cluster bins the data internally, thus providing automatic data discretization. However, if manual binning is used, the bin values should be represented by contiguous integer numbers starting at 1.

1.4.5.2 Attribute Type

Binary attributes should be declared as categorical. O-Cluster distinguishes between continuous and discrete numerical attributes. The two types of attributes undergo different binning procedures in order to capture the characteristics of the underlying distributions. For example, a discrete numerical attribute such as *age* should be declared of data type INTEGER. On the other hand, continuous numerical attributes such as height measured in feet should be declared of data type NUMBER.

1.4.5.3 Scoring

The clusters discovered by O-Cluster are used to generate a Bayesian probability model that is then used during scoring (model apply) for assigning data points to clusters. The generated probability model is a mixture model where the mixture components are represented by a product of independent normal distributions for numerical attributes and multinomial distributions for categorical attributes.

The main characteristics of the enhanced *k*-means and O-Cluster algorithms are summarized in Table 1–2, below.

Table 1–2 Comparison of Enhanced k-Means and O-Cluster

Feature	Enhanced *k*-means	O-Cluster
Clustering methodology	Distance-based	Grid-based
Number of cases	Handles tables of any size. Uses summarization for tables that don't fit in the memory buffer.	Good for data tables that have more than 1,000 cases
Number of attributes	Good for datasets that have 10 or fewer attributes	Good for data tables that have more than 10 attributes
Number of clusters	User-specified	Automatically determined
Attribute type	Numerical attributes only	Numerical and categorical attributes
Hierarchical clustering	Yes	Yes
Probabilistic cluster assignment	Yes	Yes
Automatic data normalization	Yes	Yes

1.4.6 Predictor Variance Algorithm

ODM Attribute Importance is implemented using the Predictor Variance algorithm. Predictor Variance estimates the variances of the predictor target combinations and the variance with respect to the other predictors.

The basic concept is that the higher the sum of the variances, the more informative the predictor attribute is in the build data table. These statistics give an idea of how correlated each predictor is with the target attribute. Predictor variance assesses the relative usefulness of each attribute for making predictions for rows in general, instead of making a prediction for any particular case.

For algorithms like ABN that perform automatic attribute selection, the attributes selected may differ significantly from Predictor Variance. This is because of the different techniques used to derive the important attributes.

1.4.7 Apriori Algorithm

The association rule mining problem can be decomposed into two subproblems:

- Find all combinations of items, called frequent itemsets, whose support is greater than minimum support.

- Use the frequent itemsets to generate the desired rules. The idea is that if, for example, ABCD and AB are frequent, then the rule AB implies CD holds if the ratio of support(ABCD) to support(AB) is at least as large as the minimum confidence. Note that the rule will have minimum support because ABCD is frequent.

The Apriori algorithm for finding frequent itemsets makes multiple passes over the data. In the *k*th pass, it finds all itemsets having *k* items, called the *k*-itemsets. Each pass consists of two phases. Let F_k represent the set of frequent *k*-itemsets, and C_k the set of candidate *k*-itemsets (potentially frequent itemsets). First, is the candidate generation phase where the set of all frequent (*k*-1) itemsets, F_{k-1}, found in the (*k*-1)th pass, is used to generate the candidate itemsets C_k. The candidate generation procedure ensures that C_k is a superset of the set of all frequent *k*-itemsets. A specialized in-memory hash-tree data structure is used to store C_k. Then, data is scanned in the support counting phase. For each transaction, the candidates in C_k contained in the transaction are determined using the hash-tree data structure and their support count is incremented. At the end of the pass, C_k is examined to determine which of the candidates are frequent, yielding F_k. The algorithm terminates when F_k or C_{k+1} becomes empty.

In ODM, we use an SQL-based implementation of the Apriori algorithm. The candidate generation and support counting steps are implemented using SQL queries. We do not use any specialized in-memory data structures. The SQL queries are fine-tuned to run efficiently in the database server by using various hints.

1.5 Data Mining Tasks

Data mining tasks in ODM include model building, testing, computing lift, and applying (scoring), as well as importing and exporting a PMML representation of certain models.

All models go through a build process. Classification models also have a testing phase in which a different data table also containing known target values is presented to the model and the predicted value is compared with the known (or actual) target values. Association Rules, Attribute Importance, and clustering models do not have a testing phase, nor do they compute lift. Classification models and clustering models can both be used to score a data table, whereas an association rules model does not support scoring. ODM imports and exports PMML models for Naive Bayes classification and Association Rules. Attribute Importance supports only build since it produces an importance ordering of the attributes.

Table 1–3 compares data mining tasks performed for the different ODM functions.

Table 1–3 Data Mining Tasks per Function

Function	Build	Test	Compute Lift	Apply (Score)	Import PMML	Export PMML
Classification	X	X	X	X	Naive Bayes	Naive Bayes
Clustering	X			X		
Association Rules	X				X	X
Attribute Importance	X					

1.5.1 Model Build

ODM supports two levels of settings: *function* and *algorithm*. When the function level settings do not specify particular algorithm settings, ODM chooses an appropriate algorithm and provides defaults for the relevant parameters. In general, model building at the function level eliminates many of the technical details of data mining.

Models are built in the data mining server (DMS). After a model is built, it is persisted in the DMS and can be accessed by its user-specified unique name.

The typical steps for model building are as follows:

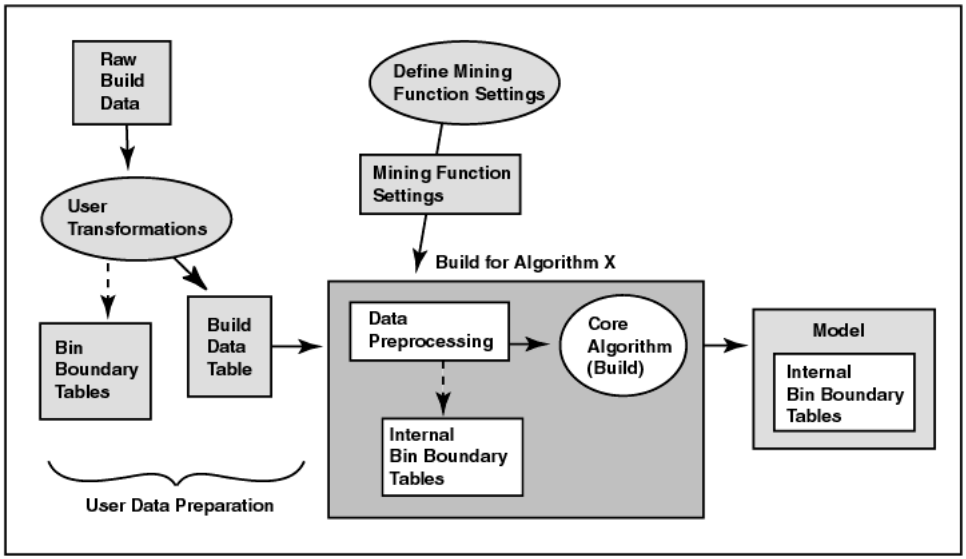
1. Specify input data by creating a physical data specification that references an existing data table or view (see Section 1.6.1, "Physical Data Specification"). This data may or may not have been prepared (for example, binned) manually (see Section 1.8, "Discretization (Binning)").
2. Create a mining function settings object, which specifies function-level parameters to the algorithm (see Section 1.6.2, "Mining Function Settings").
3. Create a logical data specification and associate it with the mining function settings (see Section 1.6.4, "Logical Data Specification").
4. Create mining algorithm settings (optional), which specifies algorithm-specific parameters to the algorithm.
5. Create a build task and invoke the execute method.

See Section 2.2, "Using ODM to Perform Mining Tasks" in Chapter 2.

Figure 1–1 illustrates the build process. Raw data undergoes the transformations specified by the user and may also be manually binned (i.e., sorted into bins) according to user-specified bin boundaries. The resulting data table, that is, the *build*

data table, is fed to the appropriate ODM algorithm, along with mining function settings. The algorithm performs further data preprocessing that may include automatic internal binning, and then performs the build. The resulting model includes bin boundary tables internal to the algorithm, i.e., the ones that resulted from automatic binning. They are not part of the model if you did not choose automatic binning.

Figure 1–1 The Build Process



1.5.2 Model Test

Classification models can be tested to get an estimate of their accuracy.

After a model is built, model testing estimates the accuracy of a model's predictions by applying it to a new data table that has the same format as the build data table (see Section 1.6.4, "Logical Data Specification"). The test results are stored in a mining test result object. A classification test result includes a *confusion matrix* (see Section 1.6.9, "Confusion Matrix") that allows a user to understand the type and number of classification errors made by the model.

1.5.3 Computing Lift

ODM supports computing lift for a classification model. Lift can be computed for binary (2 values) target fields and multiclass (more than 2 values) target fields. Given a designated positive target value, that is, the value of most interest for prediction, such as "buyer," or "has disease," test cases are sorted according to how confidently they are predicted to be positive cases. Positive cases with highest confidence come first, followed by positive cases with lowest confidence. Negative cases with lowest confidence come next, followed by negative cases with highest confidence. Based on that ordering, they are partitioned into quantiles, and the following statistics are calculated:

- *Target density* of a quantile is the number of actually positive instances in that quantile divided by the total number of instances in the quantile.
- *Cumulative target density* is the target density computed over the first *n* quantiles.
- *Quantile lift* is the ratio of target density for the quantile to the target density over all the test data.
- *Cumulative percentage of records* for a given quantile is the percentage of all test cases represented by the first *n* quantiles, starting at the end that is most confidently positive, up to and including the given quantile.
- *Cumulative number of targets* for a given quantile is the number of actually positive instances in the first *n* quantiles (defined as above).
- *Cumulative number of nontargets* is the number of actually negative instances in the first *n* quantiles (defined as above).
- *Cumulative lift* for a given quantile is the ratio of the cumulative target density to the target density over all the test data.

Targets_cumulative can be computed from the quantities that are available in the *odm_lift_result_entry* using the following formula:

*targets_cumulative = lift_cumulative * percentage_records_cumulative*

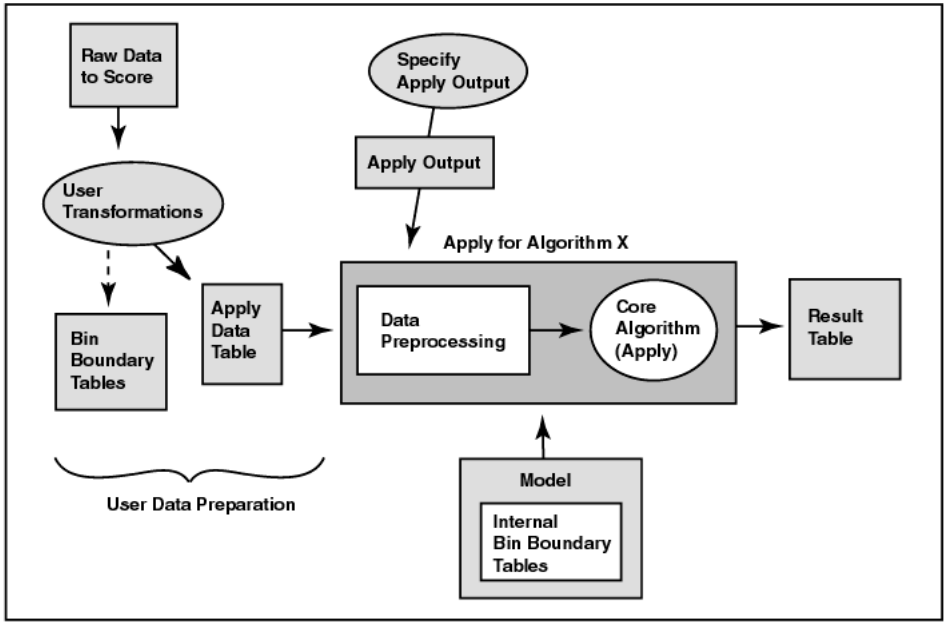
1.5.4 Model Apply (Scoring)

Applying a classification model such as Naive Bayes or Adaptive Bayes Network to data produces *scores* or *predictions* with an associated probability. Applying a clustering model to data produces, for each case, a predicted cluster identifier and the probability that the case is in that cluster. The apply data must be in the same format and state of preprocessing as the data used to build the model.

Applying a clustering model to new data produces, for each case, a predicted cluster identifier and the probability that the case belongs to that cluster. The apply data must be in the same format and state of preprocessing as the data used to build the model.

Figure 1–2 shows the apply process. Note that the input data for the apply process must undergo the same preprocessing undergone by the build data table. The data to be scored must have attributes compatible with those of the build data, that is, it must have the same attributes with the same names and respective data types or there must be a suitable mapping of one to the other. The apply data table can have attributes not found in the build data table. The result of the apply operation is placed in the schema specified by the user.

Figure 1–2 The Apply Process



The ODM user specifies the result content. For example, a user may want the *customer identifier* attribute, along with the *score* and *probability*, to be output into a table for each record in the provided mining data. This is specified using the **MiningApplyOutput** class.

ODM supports the apply operation for a table (a set of cases) or a single case (represented by a Java object). ODM supports multicategory apply, obtaining multiple class values with their associated probabilities for each case.

1.6 ODM Objects and Functionality

The principal objects that constitute Oracle9*i* Data Mining are described below.

1.6.1 Physical Data Specification

A *physical data specification* object specifies the characteristics of the physical data to be used for mining, for example, whether the data is in *transactional* or *nontransactional* format and the roles the various data columns play. The data referenced by a physical data specification object can be used in several ways: model building, testing, computing lift, scoring, transformations, etc.

ODM physical data must be in one of two formats:

- Transactional
- Nontransactional

These formats describe how to interpret each case as stored in a given database table. See Figure 1–3.

1.6.1.1 Transactional Data Format

In transactional data format, each case is stored as multiple records in a table with columns *sequenceID*, *attribute_name*, and *value* (these are names user-defined names). This format is also referred to as *multi-record case*.

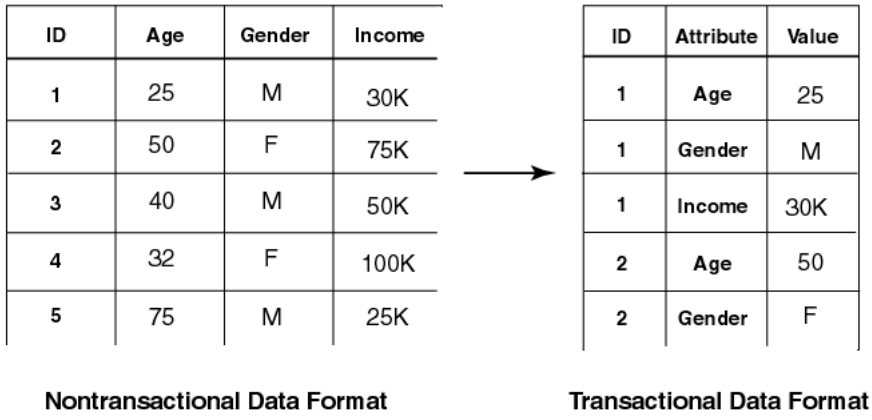
sequenceID is an integer that associates multiple records in a transactional table. *attribute_name* is a string containing the name of the attribute. *value* is an integer representing the value of the attribute. Note that the values in the Value column must be integers; they become integers when they are binned, that is, the bin ID is an integer (see Section 1.8, "Discretization (Binning)").

1.6.1.2 Nontransactional Data Format

In nontransactional format, each case is stored as one record (row) in a table. Nontransactional data is not required to provide a key column to uniquely identify each record. However, a key is recommended to associate cases with resulting scores for supervised learning. This format is also referred to as *single-record case*.

Note that in Oracle 9*i*, ODM algorithms automatically convert all nontransactional data to transactional data prior to model building. If data is already in transactional format, algorithm performance can be enhanced over data in nontransactional format.

Figure 1–3 Nontransactional and Transactional Data Format



ID	Age	Gender	Income
1	25	M	30K
2	50	F	75K
3	40	M	50K
4	32	F	100K
5	75	M	25K

Nontransactional Data Format

ID	Attribute	Value
1	Age	25
1	Gender	M
1	Income	30K
2	Age	50
2	Gender	F

Transactional Data Format

1.6.2 Mining Function Settings

A *mining function settings* (MFS) object contains the high-level parameters for building a mining model. Oracle9*i* Data Mining supports several mining algorithms for building a model.

The mining function settings allow a user to specify the type of problem to solve (for example, classification) without having to specify a particular algorithm. The ODM API allows a user to specify an algorithm; however, if none is specified, the data mining system selects an algorithm based on the function settings the user has specified. For example, if the user specifies clustering, the DMS may select *k*-means as the algorithm to build the model.

Each MFS object consists of the following:

- parameters specific to the mining function
- a logical data specification
- a data usage specification

ODM supports the persistence of mining function settings as independent, named entities in the Data Mining Server (DMS).

Table 1–4 displays function-level parameter settings and their default values.

Table 1–4 Parameter Settings by Function

Function	Parameter	Default
Classification	CostMatrix	NULL
	Priors	NULL
Clustering	MaxNumberOfClusters	20
Association Rules	MinimumSupport	0.1
	MinimumConfidence	0.1
	MaximumRuleLength	2
Attribute Importance	*None*	

1.6.3 Mining Algorithm Settings

A *mining algorithm settings* object contains the parameters associated with a particular algorithm for building a model. It allows expert data miners to fine-tune the behavior of the algorithm. Generally, not all parameters must be specified. Missing parameters are replaced with system default values. Algorithm parameters are algorithm-specific, along with their corresponding default values.

ODM's design, which separates mining algorithm settings from mining function settings, enables non-expert data miners to use ODM effectively, while expert data miners can have the control they need.

Table 1–5 displays the algorithm-level parameters and their default values. The default algorithm for a function appears in boldface type.

Table 1–5 Parameter Settings by Algorithm

Function	Algorithm	Parameter	Default
Classification	ABN	MaximumNetworkFeatureDepth	10
		MaximumNumberOfNetwork-Features	10
		MaximumConsecutivePruned-NetworkFeatures	–1
		MaximumBuildTime	NULL
	NB	singletonThreshold	0.1
		pairwiseThreshold	0.1

Table 1–5 Parameter Settings by Algorithm

Function	Algorithm	Parameter	Default
Clustering	*k*-means	DistanceFunction	Euclidean
		Iterations	7
		Error	0.05
		StoppingCriterion	ErrorAnd-Iterations
	O-Cluster	Sensitivity	0.5
Association Rules	***N/A***		
Attribute Importance	**Predictor Variance**	*None*	

1.6.4 Logical Data Specification

A *logical data specification* (LDS) object is a set of mining attribute (see Section 1.6.5, "Mining Attributes") instances that describes the logical nature of the data used as input for model building. This set of mining attributes is the basis for producing the *signature* of the model. Each mining attribute specified in a logical data specification must have a unique name.

As stored in the DMS, each MFS has its own copy of the LDS, even if references are shared in the API client process.

1.6.5 Mining Attributes

A *mining attribute* is a logical concept that describes a domain of data used as input to an ODM data mining operation. Mining attributes are either categorical or numerical. For example, domains of data include "age" ranging from 0 to 100, "buyer" with values true and false. A mining attribute specifies the name, data type, and attribute type (categorical or numeric).

1.6.6 Data Usage Specification

A *data usage specification* (DUS) object specifies how the attributes in a logical data specification (LDS) instance are used for building a model. A specification contains at most one data usage entry instance for each mining attribute in the LDS. If no data use is specified for an attribute, the default usage is active, implying that the attribute is used in building a model.

Usage includes specifying:

- whether an attribute is *active* (to be used in the model build process), *inactive* (ignored), or *supplementary* (an attribute for use with results output during scoring but not during model build)
- whether an attribute is a *target* for a supervised learning model

1.6.6.1 ODM Attribute Names and Case

ODM's treatment of attribute names differs from that of Oracle SQL. Oracle SQL can treat attribute names in a case-insensitive manner; ODM attribute names, however, are *case-sensitive*. The implications of this for ODM users are:

- The specification of attribute names must be consistent across build, test, compute lift, and apply tasks. For example, if a given target attribute name is specified for build in mixed-case format, then the same format must be maintained while specifying the attribute for test, apply, and lift.
- For a MiningApply output specification, the API allows the specification of aliases for active and supplementary attributes; the results are based on these aliases. These aliases must be unique and case-insensitive.

1.6.7 Mining Model

A *mining model* object is the result of building a model based on a mining function settings object. The representation of the model depends on the algorithm specified by the user or selected by the DMS. Some models can be used for direct inspection, for example, to examine the rules produced from association rules or clusters, others to generate predictions, for example, using a classification model.

ODM supports the persistence of mining models as independent, named entities in the DMS. A mining model contains a copy of the mining function settings (MFS) used to build it. Models cannot be stored by the user.

1.6.8 Mining Results

A *mining result* object contains the end products of one of the following mining tasks: build, test, compute lift, or apply. ODM supports the persistence of mining results as independent, named entities in the DMS.

A mining result object contains the operation start time and end time, the name of the model used, input data location, and output data location (if any) for the data mining operation.

A *build result* contains the model details. It provides the function and algorithm name of the model.

An *apply result* names the destination table (schema and table name) for the result.

A *test result,* for classification models, contains the model accuracy and references the confusion matrix.

A lift result of the lift elements is calculated on a per-quantile basis.

1.6.9 Confusion Matrix

The row indexes of a confusion matrix correspond to *actual values* observed and used for model testing; the column indexes correspond to *predicted values* produced by applying the model to the test data. For any pair of actual / predicted indexes, the value indicates the number of records classified in that pairing. For example, a value of 25 for an actual value index of "buyer" and a predicted value index of "nonbuyer" indicates that the model incorrectly classified a "buyer" as a "nonbuyer" 25 times. A value of 516 for an actual / predicted value index of "buyer" indicates that the model correctly classified a "buyer" 516 times.

The predictions were correct 516 + 725 = 1241 times, and incorrect 25 + 10 = 35 times. The sum of the values in the matrix is equal to the number of scored records in the input data table. The number of scored records is the sum of correct and incorrect predictions, which is 1241 + 35 = 1276. The error rate is 35 / 1276 = 0.0274; the accuracy rate is 1241 / 1276 = 0.9725.

A *confusion matrix* provides a quick understanding of model accuracy and the types of errors the model makes when scoring records. It is the result of a test task for classification models.

Figure 1–4 Confusion Matrix

		Predicted	
		Buyer	Non-Buyer
Actual	Buyer	516	25
	Non-Buyer	10	725

1.6.10 Mining Apply Output

A *mining apply output* instance contains several *items* that allow users to tailor the results of a model apply operation. Output can be in one or more of the following forms:

- scalar data to be passed through to the output from the input data table, for example, key attributes
- computed values from the apply itself such as score and probability
- for transactional input data, the sequence ID associated with a given case

The resulting data representation is in nontransactional form (one record per result).

Through the mining apply object, ODM supports specifying names for the resulting data columns.

There are two types of input to the apply mining operation: a *database table* for batch scoring and an *individual record* for record scoring. Apply input data must contain the same attributes that were used to build the model. However, the input data may contain additional attributes, which may appear in the output to describe the output (see source attribute, below).

Batch scoring using an input database table results in a table called the *apply output* table. An input record is represented as an instance of *RecordInstance* that contains a set of *AttributeInstance* objects, each of which describes the name of the attribute, the data type, and the value. The result of record scoring is also an instance of *RecordInstance*. The output of the apply mining operation is specified by *MiningApplyOutput*.

An instance of *MiningApplyOutput* is a specification of the data to be included in the apply output (either a table or a record) created as the result of the *apply* mining operation. The columns (or attributes) in the apply output are described by a combination of multiple *ApplyContentItem* objects. Each item can be one of the following:

- **Source attribute:** The apply output table (or record) may contain columns copied directly from the input table (or record). These are called *source attributes*, and each is represented by an instance of *ApplySourceAttributeItem*. Source attributes can be used to identify the individual source cases in the apply output, i.e., associate a key with each output record. There can be no more than 997 source attributes the output table.
- **Multiple predictions based on probability:** An instance of *ApplyMultipleScoringItem* results in top or bottom *n* predictions ordered by probability of the predictions, where *n* can range from 1 to the total number of

target values. One such item produces two columns in the output: prediction and probability, each of which is named by the user. There can be at most one instance of *ApplyMultipleScoringItem* in a *MiningApplyOutput* object.

Typically, users select "top" with $n = 1$ for obtaining the top likely prediction for each case from, for example, a classification model. However, other users may require seeing the top three predictions, for example, for recommending products to a customer.

- **Multiple predictions based on target values**: An instance of *ApplyTargetProbabilityItem* results in predictions for target values. Each such target value must be one of the original target values used to build the model. A given target value can be specified at most once. One such item produces up to three columns in the output: prediction, (optional) probability, and (optional) rank, each of which is named by the user. Probability and rank are optional. There can be at most one instance of *ApplyTargetProbabilityItem* or *ApplyMultipleScoringItem* in a *MiningApplyOutput* object. This option is useful when interested in the probability of a particular prediction, for example, if a retailer has many red sweaters, what is the probability the customer would buy something red?

The number of columns in the apply output table varies depending on the combination of items. When multiple target values are specified by *MiningApplyOutput* (if $n > 1$), n rows of output table correspond to the prediction for an input row.

Consider an input table of 15 rows. If the top 2 predictions ($n = 2$) with probabilities are specified in *MiningApplyOutput* with one source attribute from the input table, there will be 3 columns in the output table: the source attribute, the prediction, and its probability.The number of rows in the output table is 30 because the result of apply for each input row will be 2 rows (top 2) in the output table.

If the input data is transactional, the sequence ID is automatically included in the output table. However, explicit inclusion of source attributes is required for nontransactional data.

1.7 Missing Values

In this section, we discuss ODM's handing of missing values.

1.7.1 Missing Values Handling

Data tables often contain missing values, that is, one or more of the attributes in a case have a null value. ODM handles missing values as follows, depending on the data format:

- For nontransactional data, an attribute name and value pair is used only if the value is not null; otherwise, it is ignored.
- For transactional data, the row is ignored if either the sequence ID, attribute name, or value is null.

Note that this means it is the empty "cell" that is ignored, not the entire record. By "cell", we mean the intersection of one row and one column (attribute name). For transactional data, it can be said that the "row" is ignored because in transactional format, each row contains the value for one attribute.

1.8 Discretization (Binning)

ODM algorithms require that input data be *discretized* (*binned*) before model building, testing, computing lift, and applying (scoring). Binning means grouping related values together to reduce the number of distinct values for an attribute. Having fewer bins typically leads to a more compact model and one that builds faster, but it can also lead to some loss in accuracy. The target attribute is typically not binned.

The decisions about the number of bins, the values that are to be assigned to each bin, where the bin boundaries are to be set, etc., can be made according to program defaults or can be specified by the user.

The best 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 of the data and problem, ODM can bin the data using default settings.

1.8.1 Numerical and Categorical Attributes

The distinction between categorical and numerical attributes depends on the application, that is, the application determines how to treat a particular attribute. The main difference is whether the values of an attribute have a certain order.

Values of a categorical attribute do not have any meaningful order; values of a numerical attribute do have a meaningful order. This does not mean that the values of a categorical attribute cannot be ordered, but rather that the order does not matter. For example, since U.S. postal codes are numbers, they can be ordered; however, their order is not necessarily meaningful to an application, and they can therefore be considered categorical.

1.8.2 Automated Binning

Because binning can have an effect on a model's accuracy, it is better when an expert specifies the binning based on some information about the data or the problem. However, in cases where there is no additional information to inform binning decisions or when the purpose is to get an initial understanding of the problem, it makes sense to use some form of automatic binning that uses global defaults. For example, group all attributes into five bins, where five is a globally specified default. Then for categorical attributes, the five most frequent values are assigned to five different bins, and all remaining values are assigned to a sixth bin.

For numerical attributes, the values are divided into five groups of equal size according to their order.

An important advantage of automated binning is that it allows ODM to handle raw (unprepared) data, and thus simplifies what can be a daunting task for the user new to data mining. Automated binning also allows initial exploration of problems about which there is little or no information to guide binning decisions.

1.8.3 Data Preparation

ODM provides four ways to bin data:

- Explicit specification: The user provides the bin boundaries for one or more attributes. The user provides one of the following:
  - for categorical data, a list of categorical values to be contained in each bin
  - for numerical data, a set of upper and lower boundaries for the bins
- Top N most frequent items: For categorical attributes only, the user selects the value N and the name of the "other" category. ODM determines the N most frequent values and puts all other values in the "other" category.
- Quantile binning: For numerical attributes only, the values are sorted, and the values are divided into the number of user-specified quantiles. ODM determines which values are in which bins.

- Automated binning: In cases where the user has no insight into how to define optimal bins or needs to get an initial understanding of the problem, ODM can perform the binning.

An example of binning is shown below. Table 1–6 displays original data, before binning. Table 1–7 shows the bin boundaries for numeric data; Table 1–8 shows bin boundaries for categorical data. Table 1–9 shows the results of binning.

Table 1–6 Binning Illustration: Data before Binning

PERSON_ID	AGE	WORK CLASS	WEIGHT	EDUCATION	MARITAL_ STATUS	OCCUPATION
2	27	Private	160972	HS-grad	Married	Crafts
8	46	Private	116635	Bach.	Separ.	Prof.
10	34	Private	62124	HS-grad	Separ.	Agricultural
11	23	Sta-gov	103588	< Bach.	NeverM	Cleric.
41	30	Private	178835	< Bach.	Married	Sales

Table 1–7 Binning Illustration: Bin Boundaries for Numeric Data

COLUMN_ NAME	LOWER_ BOUNDARY	UPPER_ BOUNDARY	BIN_ID	IS_ ATTRIBUTE	DISPLAY_NAME
AGE	17	24.3	1	T	17-24.3
AGE	24.3	31.6	2	T	24.3-31.6
AGE	31.6	38.9	3	T	31.6-38.9
AGE	38.9	46.2	4	T	38.9-46.2
AGE	46.2	53.5	5	T	46.2-53.5
WEIGHT	13769	122137.4	1	T	13769-122137.4
WEIGHT	122137.4	230505.8	2	T	122137.4-230505.8
WEIGHT	230505.8	338874.2	3	T	230505.8-338874.2
WEIGHT	338874.2	447242.6	4	T	338874.2-447242.6

Table 1–8 Binning Illustration: Bin Boundaries for Categorical Data

COLUMN_NAME	CATEGORY	BIN_ID	DISPLAY_ NAME	IS_ ATTRIBUTE
WORKCLASS	Loc-gov	100	Governme nt	T
WORKCLASS	Fed-gov	100	Governme nt	T
WORKCLASS	Sta-gov	100	Governme nt	T
WORKCLASS	Private	300	Others	T
EDUCATION	HS-grad	1	HS-grad	T
EDUCATION	< Bach.	2	< Bach.	T
EDUCATION	Bach.	3	Bach.	T
EDUCATION	Masters	4	Masters	T
MARITAL_ STATUS	Married	1	Married	T
MARITAL_ STATUS	NeverM	2	NeverM	T
MARITAL_ STATUS	Divorc.	3	Divorc.	T
MARITAL_ STATUS	Widowed	4	Widowed	T
MARITAL_ STATUS	Separ.	5	Separ.	T
OCCUPATION	Prof	1	Prof	T
OCCUPATION	Crafts	2	Crafts	T
OCCUPATION	Exec.	3	Exec.	T
OCCUPATION	Sales	4	Sales	T
OCCUPATION	Cleric	5	Cleric	T
OCCUPATION		6	Other_occ	T

Table 1–9 Binning Illustration: Assignment of Original Data to Bins

PERSON_ID	AGE	WORK CLASS	WEIGHT	EDUCATION	MARITAL_ STATUS	OCCUPATION
2	2	300	2	1	1	2
8	4	300	1	3	5	1
10	3	300	1	1	5	6
11	1	100	1	2	2	5
41	2	300	2	2	1	4

Note: 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.

1.9 PMML Support

The Predictive Model Markup Language (PMML) specifies data mining models using an XML DTD (document type definition). PMML provides a standard representation for data mining models to facilitate model interchange among vendors. PMML is specified by the Data Mining Group (http://www.dmg.org).

ODM is both a producer and consumer of PMML models. That is, ODM can produce (generate) a PMML model that can be used by other software that can consume PMML. ODM can also consume PMML models, that is, ODM can convert certain PMML model representations to valid ODM models. ODM is a producer and consumer of two model types: Association Rules models and Naive Bayes classification models.

ODM consumes only models that use features supported by ODM.

2

ODM Programming

This chapter discusses two major topics:

- The requirements for compiling and executing ODM programs
- How to perform common data mining tasks using Oracle9*i* Data Mining (ODM).

For an example of ODM basic usage, see Chapter 3.

This chapter provides an overview of the steps required to perform basic ODM tasks. For detailed examples of how to perform these tasks, see the ODM sample programs. The ODM sample programs are distributed with the ODM documentation. For an overview of the ODM sample programs, see Appendix A.

This chapter does not include a detailed description of any of the ODM API classes and methods. For detailed information about the ODM API, see the ODM Javadoc in the directory `$ORACLE_HOME/dm/doc` on any system where ODM is installed.

2.1 Compiling and Executing ODM Programs

ODM depends on the following Oracle9*i* Java Archive (`.jar`) files:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqapi.jar
$ORACLE_HOME/rdbms/jlib/xsu12.jar
$ORACLE_HOME/dm/lib/odmapi.jar
```

These files must be in your CLASSPATH to compile and execute ODM programs.

If you use a database character set that is *not* US7ASCII, WE8DEC, WE8ISO8859P1, or UTF8, you must also include the following in your CLASSPATH:

```
$ORACLE_HOME/jdbc/lib/nls_charset12.zip
```

If you do not include `nls_charset12.zip` in your CLASSPATH, an ODM program will fail with the following error:

```
oracle.jms.AQjmsException: Non supported character set:oracle-character-set-178
```

2.2 Using ODM to Perform Mining Tasks

This section describes the steps required to perform several common data mining tasks using ODM.

All work in ODM is done using `MiningTask` objects.

2.2.1 Build a Model

This section summarizes the steps required to build a model.

1. Prepocess the input data, as required.
2. Discretize (bin) the input data. (This step is optional, ODM algorithms can automatically bin input data.)
3. Construct and store a `MiningFunctionSettings` object.
4. Construct and store a `MiningBuildTask` object.
5. After successful construction of the build task object, call a store method to store the object in the data mining server.
6. Call the execute method; the execute method queues the work for asynchronous execution and returns a task identifier to the caller.
7. 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 build results object exists.

The following sample programs illustrate building ODM models:

- `Sample_AdaptiveBayesNetworkBuild.java`
- `Sample_NaiveBayesBuild.java`
- `Sample_AssociationRulesBuild.java`
- `Sample_ClusteringBuild.java`

2.2.2 Perform Tasks in Sequence

Data mining tasks are usually performed in sequence. The following sequence of tasks is typical:

1. Collect and preprocess data
2. Build a model
3. Test the model
4. Calculate lift
5. Apply the model

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:

1. Perform the build task as described in Section 2.2.1 above.
2. After successful completion of the build task, start the test task by calling the execute method on a `MiningTestTask` object. Either periodically check the status of the test operation or block until the task completes.
3. After successful completion of the test task, execute the compute lift task by calling the execute method on a `MiningComputeLiftTask` object.

2.2.3 Find the Best Model

Model Seeker builds multiple models; it then evaluates and compares the models to find a "best" model.

Follow these steps to use Model Seeker:

1. Create a single `ModelSeekerTask` (MST) instance to hold the information needed to specify the models to build. The required information is defined in subclasses of the `MiningFunctionSettings` (MFS) and `MiningAlgorithmSettings` (MAS) classes.

   You can specify a combination of as many instances of the following as desired:

   - `NaiveBayesAlgorithmnSettings`
   - `CombinationNaiveBayesSettings`

- `AdaptiveBayesNetworkAlgorithmSettings`
- `CombinationAdaptiveBayesNetSettings`

(You cannot specify clustering models or Association Rules models.)

2. Call the Model Seeker Task execute method. The method returns once the task is queued for asynchronous execution.
3. Periodically call the `getCurrentStatus` method to get the status of the task, using the task name. Alternatively, use the `waitForCompletion` method to wait until all asynchronous activity for the required work completes.
4. When the model seeker task completes, use the `getResults` method to view the summary information and the best model. Model Seeker discards all models that it builds except the best one.

The sample program `Sample_ModelSeeker.java` illustrates how to use Model Seeker.

2.2.4 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 these attributes only.

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 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.

The sample program `Sample_AttributeImportanceBuild.java` illustrates how to build an attribute importance model.

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 `MiningFunctionSetting`. Only the attributes returned by Attribute Importance will be active for model building.

3. Build a model using the new Logical Data Specification and Data Usage Specification.

The sample program `Sample_AttributeImportanceUsage.java` illustrates how to build a model using the important attributes.

2.2.5 Apply a 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 transactional format, any table that you apply that model to must be in transactional format. Similarly, if the table used to build the model was in nontransactional 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.

3

ODM Basic Usage

This chapter contains complete examples of using ODM to build a model and then score new data using that model. These examples illustrate the steps that are required in all code that uses ODM. The following two sample programs are discussed in this chapter:

- `Sample_NaiveBayesBuild_short.java` (Section 3.2)
- `Sample_NaiveBayesApply_short.java` (Section 3.3)

The complete code for these examples is included in the ODM sample programs that are installed when ODM is installed. For an overview of the ODM sample programs, see Appendix A. For detailed information about compiling and linking these programs, see Section A.6.

The data that the sample programs use are included with sample programs: `Sample_NaiveBayesBuild_short.java` uses `census_2d_build_unbinned` and `Sample_NaiveBayesApply_short.java` uses `census_2d_apply_unbinned`. For more information about these tables, see Section A.4. The data used sample programs is installed in the `ODM_MTR` schema.

This chapter does not include a detailed description of any of the ODM API classes and methods. For detailed information about the ODM API, see the ODM Javadoc in the directory `$ORACLE_HOME/dm/doc` on any system where ODM is installed.

The sample programs have a number of steps in common. Common steps are repeated for simplicity of reference.

These short sample programs use data tables that are used by the other ODM sample programs. The short sample program that builds a model uses the table `CENSUS_2D_BUILD_UNBINNED`; the short sample program that applies the model uses `CENSUS_2D_APPLY_UNBINNED`. For more information about these tables, see Section A.4.

Note that these short sample programs do not use the property files that the other ODM use.

Note: If you execute `SampleNaiveBayesBuild.java` and then execute `SampleNaiveBayesBuild_short.java` or `Sample_NaiveBayesApply_short.java`, you must change the `buildtablename` to a new name. Otherwise, you get a unique constraint error because the model name, MFS name, and Mining Task name are identical in both programs.

3.1 Using the Short Sample Programs

The short sample programs must be compiled and then executed in the proper order; you must execute `SampleNaiveBayesBuild_short.java` (which builds the model) before you execute `SampleNaiveBayesApply_short.java` (which applies the model to data).

You can compile and execute these models in several ways:

- Using scripts provided with the sample programs
- Using Oracle9*i* JDeveloper

These methods are described in Section A.6.

Note that the short sample programs do not use property files.

3.2 Building a Model

This section describes the steps that must be performed by any program that builds an ODM model.

The sample program `Sample_NaiveBayesBuild_short.java` is a complete executable program that illustrates these required steps. The data for the sample program is `CENSUS_2D_BUILD_UNBINNED`.

3.2.1 Before Building an ODM Model

Before you build an ODM model, ODM must be installed on your system. You need to know the URL of the database where the ODM Data Mining Server resides, the user name, and the password. (Ask the person who installed the program what the user name and password are.)

Before you execute an ODM program, the ODM Monitor must be running.

Before you build a model, you must identify the data to be used during model building. The data must reside in a table in an Oracle9*i* database. You should clean the data as necessary; for example, you may want to treat missing values and deal with outliers, that is, extreme values that are either errors or values that may skew the binning. The table that contains the data can be in either transactional or nontransactional form. The ODM sample programs include data tables to use for model building.

Before you building a model, you must also know what data mining function that you wish to perform; for example, you may wish to create a classification model. You may specify which algorithm to use or let ODM decide which algorithm to use. The sample programs described in this chapter build and apply a Naive Bayes model.

3.2.2 Main Steps in ODM Model Building

For ODM to build a model, ODM must know the answers to the following questions:

- Which server should be used to do the mining?
- Where is the data for mining and how is it organized?
- What type of model should be built? What is its function? Which algorithm should be used?

The following steps provide answers to the questions asked above:

1. Connect to the DMS (data mining server).
2. Create a `PhysicalDataSpecification` object for the build data.
3. Create a `MiningFunctionSettings` object (in this case, a `ClassificationFunctionSettings` object with no supplemental attributes).
4. Build the model.

The steps are illustrated below with code for building a Naive Bayes model.

3.2.3 Connect to the Data Mining Server

Before building a model, it is necessary to create an instance of `DataMiningServer`. This instance is used as a proxy to create connections to a Data Mining Server (DMS). The instance also maintains 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.

```
//Create an instance of the DMS server.
//The mining server DB_URL, user_name, and password for your installation
//need to be specified
dms=new DataMiningServer("DB_URL", "user_name", "password");

//get the actual connection
dmsConnection = dms.login();
```

3.2.4 Describe the Build Data

Before ODM can use data to build a model, it must know where the data is and how the data is organized. This is done through a `PhysicalDataSpecification` instance where you indicate whether the data is in nontransactional or transactional format and describe the roles the various data columns play.

3.2.4.1 Location Access Data for Build Data

Before you create a `PhysicalDataSpecification` instance, you must provide information about the location of the build data. This is accomplished using a `LocationAccessData` object.

```
//Create a LocationAccessData using the table_name
//(CENSUS_2D_BUILD_UNBINNED) and schema_name for your installation
LocationAccessData lad =
     new LocationAccessData("CENSUS_2D_BUILD_UNBINNED", "schema_name");
```

Next, create the actual `PhysicalDataSpecification` instance.

3.2.4.2 Physical Data Specification for Nontransactional Build Data

If the data is in nontransactional format, all the information needed to build a `PhysicalDataSpecification` is contained in the `LocationAccessData` object.

```
//Create the actual PhysicalDataSpecification for a
//NonTransactionalDataSpecification object since the
//data set is nontransactional
PhysicalDataSpecification m_PhysicalDataSpecification =
                  new NonTransactionalDataSpecification(lad);
```

3.2.4.3 Physical Data Specification for Transactional Build Data

If the data is in transactional format, you must specify the role that the various data columns play.

```
//Create the actual PhysicalDataSpecification for a transactional
//data case
PhysicalDataSpecification m_PhysicalDataSpecification =
        new TransactionalDataSpecification(
                "CASE_ID",     //column name for sequence id
                "ATTRIBUTES",  //column name for attribute name
                "VALUES",      //column name for value
                lad);
```

3.2.5 Create the MiningFunctionSettings Object

The `MiningFunctionSettings` (MFS) object tells the DMS the type of model to build, the function of the model, and the algorithm to use.

ODM supports the following mining functions:

- Association rules (unsupervised learning)
- Clustering (unsupervised learning)
- Classification (supervised learning)
- Attribute importance (supervised learning)

The MFS allows a user to specify the type of result desired without having to specify a particular algorithm. If an algorithm is not specified, the underlying DMS selects the algorithm based on user-provided parameters.

3.2.5.1 Specify the Default Algorithm for Classification

To build a model for classification using ODM's default classification algorithm, use a `ClassificationFunctionSettings` object with a null `MiningAlgorithmSettings` for the MFS. An easy way to create a `ClassificationFunctionSettings` object is to use the `create` method, as illustrated below. In this case, it is necessary to indicate the name of the target attribute, the type of the target attribute, and whether the data has been prepared (binned) by the user. Unprepared data will automatically be binned by ODM.

```
//Specify "class" as the target attribute name, categorical for the target
//attribute type, and set the DataPreparationStatus to unprepared.
//Automatic binning will be applied in this case.
ClassificationFunctionSettings m_ClassificationFunctionSettings =
```

```
ClassificationFunctionSettings.create(
         dmsConnection,
         null,
         m_PhysicalDataSpecification,
         "class",
         AttributeType.categorical,
         DataPreparationStatus.getInstance("unprepared"));
```

3.2.5.2 Specify the Naive Bayes Algorithm

If a particular algorithm is to be used, the information about the algorithm is captured in a `MiningAlgorithmSettings` instance. For example, if you want to build a model for classification using the Naive Bayes algorithm, first create a `NaiveBayesSettings` instance to specify settings for the Naive Bayes algorithm. Two settings are available: `singleton threshold` and `pairwise threshold`. Then create a `ClassificationFunctionSettings` instance for the build operation.

```
//Create the Naive Bayes algorithm settings by setting the thresholds
//to 0.01.
NaiveBayesSettings algorithmSetting = new NaiveBayesSettings(0.01f 0.01f);

//Create the actual ClassificationFunctionSettings using
//algorithmSetting for MiningAlgorithmSettings. Specify "class" as
//the target attribute name, "categorical" for the target attribute
//type, and set the DataPreparationStatus to "unprepared".
//Automatic binning will be applied in this case.
ClassificationFunctionSettings m_ClassificationFunctionSettings =
          ClassificationFunctionSettings.create(
                 dmsConnection,
                 algorithmSetting,
                 m_PhysicalDataSpecification,
                 class,
                 Attribute Type.categorical,
                 DataPreparationStatus.getInstance(unprepared));
```

3.2.5.3 Validate the Mining Function Settings for Build

Because `MiningFunctionSettings` objects are complex objects, it is good practice to validate whether they were correctly created before starting the actual build task. If the `MiningFunctionSettings` object is a valid one, it should be persisted in the DMS for later use. This is illustrated below for the `ClassificationFunctionSettings` in our example.

```
//Validate and store the ClassificationFunctionSettings object
//with the name "Sample_NB_MFS".
m_ClassificationFunctionSettings.validate();
m_ClassificationFunctionSettings.store(dmsConnection, "Sample_NB_MFS");
```

3.2.6 Build the Model

Now that all the required information for building the model has been captured in an instance of `PhysicalDataSpecification` and `MiningFunctionSettings`, the last step needed is to decide whether the model should be built synchronously or asynchronously.

If you are calling ODM from an application, the design of the calling application may determine whether to build the model synchronously or asynchronously. Also, if the data used to build the model is large, it may take a significant amount of time to build the model; in such a case, you will probably want to build the model asynchronously.

3.2.6.1 Build the Model Synchronously

For a synchronous build, use the static `MiningModel.build` method. Note that this method is deprecated for ODM release 2.

```
//Build the model using the MFS named "Sample_NB_MFS" and store the
//model under the name "Sample_NB_Model".
MiningModel.build(
     dmsConn,
     lad,
     m_PhysicalDataSpecification,
     "Sample_NB_MFS",
     "Sample_NB_Model");
```

3.2.6.2 Build the Model Asynchronously

For an asynchronous build, create an instance of `MiningTask`. A mining task can be persisted in the DMS using the `store` method and executed at any time; however, it can be executed only once. Once the task is executing, query the current status information of a task by calling the `getCurrentStatus` method. This call returns a `MiningTaskStatus` object, which provides more details about the state. You can get the complete status history of a task by calling the `getStatusHistory` method.

```
//Create a Naive Bayes build task and execute it.
//MiningFunctionsSettings name (for example, "Sample_NB_MFS"), and
//the ModelName (for example, "Sample_NB_Model") need to be specified.
```

```
MiningBuildTask task =
     new MiningBuildTask(
          m_PhysicalDataSpecification,
          "Sample__NB_MFS",
          "Sample_NB_Model");

 //Store the task under the name "Sample_NB_Build_Task"
task.store(dmsConnection, "Sample_NB_Build_Task");

//Execute the task
task.execute(dmsConnection);
```

After the `MiningModel.build` or the `task.execute` call successfully completes, the model will be stored using the name that you specified (in this case, `Sample_NB_Model`) in the DMS.

3.3 Scoring Data Using a Model

After you've created a model, you can apply it to new data to make predictions; the process is referred to as "scoring data."

ODM can be used to score multiple records specified in a single database table or to score a single record. This section describes scoring multiple records.

The sample program `Sample_NaiveBayesApply_short.java` is a complete executable program that illustrates these required steps. The data for this sample program is `CENSUS_2D_APPLY_UNBINNED`. Note that this sample program does not use a property file.

3.3.1 Before Scoring Data

Before scoring an ODM model, you must have built an ODM model. This implies that ODM is installed on your system, and that you know the location of the database, the user name, and the password.

Before executing an ODM program, the ODM Monitor must be running.

Before you score data, the data must reside in a table in an Oracle9*i* database. The data to score must be compatible with the build data that you used when you built the model. You should clean the data to be scored in the same way that you cleaned the build data. If the build data for the model was not binned, the data to score must also be not binned.

The table that contains the data to score can be in either transactional or nontransactional form.

3.3.2 Main Steps in ODM Scoring

For ODM to score data using a model, ODM must know the answers to the following questions:

- Which server should be used to do the scoring?
- Where is the data for scoring and how is it organized?
- Where should the output be stored?
- What information do you want returned as the result of scoring?

The following steps provide answers to the questions above:

1. Connect to the DMS (data mining server).
2. Create a `PhysicalDataSpecification` object for the input data (the data that you want to score).
3. Create a `LocationAccessData` object for the input and output data.
4. Create a `MiningApplyOutput` object for the output data.
5. Score the data.

The steps above are illustrated in this section with code for scoring a Naive Bayes model.

3.3.3 Connect to the Data Mining Server

Before scoring data, it is necessary to create an instance of `DataMiningServer`. This instance is used as a proxy to create connections to a Data Mining Server (DMS). The instance also maintains 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.

```
//Create an instance of the DMS server.
//The mining server DB_URL, user_name, and password for your installation
//need to be specified.
dms=new DataMiningServer("DB_URL", "user_name", "password");

//get the actual connection
```

```
dmsConnection = dms.login((();
```

3.3.4 Describe the Input Data

Before ODM can apply a model to data, it must know the physical layout of the data. This is done through a `PhysicalDataSpecification` instance where you indicate whether the data is in nontransactional or transactional format and describe the roles the various data columns play.

3.3.4.1 Location Access Data for Apply Input

Before you create a `PhysicalDataSpecification` instance, you must provide information about the location of the input data. This is accomplished using a `LocationAccessData` object.

```
//Create a LocationAccessData using the table_name
//(CENSUS_2D_APPLY_UNBINNED) and the schema_name for your installation
LocationAccessData lad =
    new LocationAccessData("CENSUS_2D_APPLY_UNBINNED", "schema_name)";
```

Next, create the `PhysicalDataSpecification` instance.

3.3.4.2 Physical Data Specification for Nontransactional Input Data

If the data is in nontransactional format, all the information needed to build a `PhysicalDataSpecification` is contained in the `LocationAccessData` object.

```
//Create the actual PhysicalDataSpecification for a
//NonTransactionalDataSpecification object since the
//data set is nontransactional
PhysicalDataSpecification m_PhysicalDataSpecification =
                new NonTransactionalDataSpecification(lad);
```

3.3.4.3 Physical Data Specification for Transactional Input Data

If the data is in transactional format, you must specify the role that the various data columns play.

```
//Create the actual PhysicalDataSpecification for transactional
//data case
PhysicalDataSpecification m_PhysicalDataSpecification =
        new TransactionalDataSpecification(
                "CASE_ID",     //column name for sequence id
                "ATTRIBUTES",  //column name for attribute name
                "VALUES",      //column name for value
```

```
                                lad);
```

3.3.5 Describe the Output Data

Before scoring the input data the DMS needs to know where to store the output of the scoring.

3.3.5.1 Location Access Data for Apply Output

Create a `LocationAccessData` object specifying where to store the apply output. The following code specifies writing to the output table `CENSUS_NB_APPLY_RESULT`.

```
// LocationAccessData for output table to store the apply results.
LocationAccessData ladOutput = new LocationAccessData ("CENSUS_NB_APPLY_RESULT",
                                    "output_schema_name");
```

3.3.6 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 are involved in creating a `MiningApplyOutput` object:

1. Create an empty `MiningApplyOutput` object.
2. Create an `ApplyContentItem` object describing which generated columns to be included in the output and add it to the `MiningApplyOutput` object.
3. Create `ApplyContentItem` objects describing columns from the input table to be included in the output and add them to the `MiningApplyOutput` object.
4. Validate the `MiningApplyOuput` that you created.

3.3.6.1 Create an Empty Mining Apply Output Object

Create an empty `MiningApplyOutput` object as follows:

```
// Create MiningApplyOutput object
MiningApplyOutput m_MiningApplyOutput = new MiningApplyOutput();
```

3.3.6.2 Specify the Generated Columns in the Apply Output

There are two options for generated columns, described by the following `ApplyContentItem` subclasses:

- `ApplyMultipleScoringItem`: used for generating a list of top or bottom *n* predictions ordered by their associated target value probability
- `ApplyTargetProbabilityItem`: used for generating a list of probabilities for particular target values

For the current example, let's use an `ApplyTargetProbabilityItem` instance. Before creating an instance of `ApplyTargetProbabilityItem`, it is necessary to specify the names and the data types of the prediction, probability, and rank columns for the output. This is done through `Attribute` objects.

```
// Create Attribute objects that specify the names and data
// types of the prediction, probability, and rank columns for the
// output.
Attribute predictionAttribute =
new Attribute("myprediction", DataType.stringType);
Attribute probabilityAttribute =
new Attribute("myprobability", DataType.stringType);
Attribute rankAttr =
new Attribute("myrank", DataType.stringType);

// Create the ApplyTargetProbabilityItem instance
ApplyTargetProbabilityItem aTargetAttrItem =
new ApplyTargetProbabilityItem(predictionAttribute, probabilityAttribute,
                           rankAttr);
```

An `ApplyTargetProbabilityItem` class contains a set of target values whose prediction and probability appear in the apply output table, regardless of their ranks. A target value is represented as a Category, and it must be one of the target values in the target attribute used when building the model to be applied. This step is not necessary for the `ApplyMultipleScoringItem` case.

```
// Create Category objects to represent the target values
// to be included in the apply output table. In this example
// two target values are specified.
Category target_category = new Category(″positive_class", "0",
                                  DataType.getInstance("int"));
Category target_category1 = new Category("positive_class", "1",
                                    DataType.getInstance("int"));

// Add the target values to the ApplyTargetProbabilityItem
```

```
// instance
aTargetAttrItem.addTarget(target_category);
aTargetAttrItem.addTarget(target_category1);

// Add the ApplyTargetProbabilityItem to the MiningApplyOutput
// object
m_MiningApplyOutput.addItem(aTargetAttrItem);
```

3.3.6.3 Specify the Input Columns to be Included in Output

The input table columns to be included in the apply output are described by `ApplySourceAttributeItem` instances. Each instance maps a column in the input table to a column in the output table. These columns are described by a source Attribute and a destination Attribute.

```
// In this example, attribute "PERSON_ID" from the source table
// will be returned in the column "ID" in the output table.
// This specification is captured by the
// m_ApplySourceAttributeItem object.
MiningAttribute sourceAttribute = new MiningAttribute(
                 "PERSON_ID",
                  DataType.intType,
                  AttributeType.notApplicable,
                  false,
                  false);

Attribute destinationAttribute = new Attribute(
                "ID",
                DataType.intType);

ApplySourceAttributeItem m_ ApplySourceAttributeItem =
             new ApplySourceAttributeItem(
             sourceAttribute,
             destinationAttribute)

// Add the ApplySourceAttributeItem object
// to the MiningApplyOutput object
m_MiningApplyOutput.addItem(m_ApplySourceAttributeItem);
```

Note that the source mining attribute could have been taken from the logical data of the model's function settings.

3.3.6.4 Validate the Mining Apply Output Object

Because `MiningApplyOutput` objects are complex objects, it is a good practice to validate that they were correctly created before you do the actual scoring. This is illustrated below for the `MiningApplyOutput` in our example.

```
// Validate the MiningApplyOutput
m_ MiningApplyOutput.validate();
```

3.3.7 Apply the Model

Now that all the required information for scoring the model has been captured in instances of `PhysicalDataSpecification`, `LocationAccessData`, and `MiningApplyOutput`, the last step is

- Specify how to score the data (synchronously or asynchronously)
- Tell the DMS which model to use for scoring

If you are calling ODM from an application, the design of the calling application may determine whether to apply the model synchronously or asynchronously. Also, if the input data has many cases, the apply operation may require a significant amount of time; in such a case,, you will probably want to apply the model asynchronously.

3.3.7.1 Apply the Model Synchronously

For synchronous apply, use the static `SupervisedModel.Apply` method. Note that this method is deprecated for ODM release 2.

```
// Synchronous Apply
// Score the model using the model named "Sample_NB_Model" and
// store the results in the "Sample_NB_APPLY_RESULT"
public static void apply(
              dmsConn,
              lad,
              m_PhysicalDataSpecification,
              "Sample_NB_Model",
              m_MiningApplyOutput,
              ladOutput,
              "Sample_NB_APPLY_RESULT")
```

3.3.7.2 Apply the Model Asynchronously

For asynchronous apply, it is necessary to create an instance of `MiningTask`. A mining task can be persisted in the DMS using the `store(dmsConn, taskName)` method and executed at any time; such a task can be executed only once. The current status information of a task can be queried by calling the `getCurrentStatus(dmsConn, taskName)` method. This returns a `MiningTaskStatus` object, which provides more details about the state. You can get the complete status history of a task by calling the `getStatusHistory(dmsConn, taskName)` method.

```
// Asynchronous Apply
/ Create a Naive Bayes apply task and execute it.
// Result name (e.g., "Sample_NB_APPLY_RESULT"), and the
// model name (e.g., "Sample_NB_Model") need to be specified
MiningApplyTask task =  new MiningBuildTask(
                        m_PhysicalDataSpecification,
                        "Sample_NB_Model",
                        m_MiningApplyOutput,
                        ladOutput,
                        "Sample_NB_APPLY_RESULT");

// Store the task under the name "Sample_NB_APPLY_Task"
task.store(dmsConnection, "Sample_NB_APPLY_Task");

// Execute the task
task.execute(dmsConnection);
```

A

ODM Sample Programs

Oracle9*i* Data Mining (ODM) includes sample input data and sample Java programs that illustrate techniques available with ODM's Java API. This appendix contains instructions for compiling and executing these sample programs.

The input data used by the sample programs is created as part of the ODM install; the data is available in the `odm_mtr` schema.

You can compile and execute the sample programs using provided scripts; you can also compile and execute the sample programs in JDeveloper using a JDeveloper project that you can download from Oracle Technology Network (`http://otn.oracle.com`). For information about installing and using the JDeveloper project for ODM, see the README that is included in the download.

A.1 Overview of the ODM Sample Programs

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

- Data preparation (binning and attribute importance),
- Model creation
- Model testing
- Application of the model to new data (scoring the data)

Data mining models are either supervised or unsupervised.

A supervised model is used to predict the value of a designated variable, called a target, together with the confidence associated with each prediction. Supervised models are illustrated in the sample programs for Naive Bayes (NB) and Adaptive Bayes Networks (ABN).

An unsupervised model has no target variable, but is used to predict group membership or relationships of an individual. Unsupervised models are illustrated in the sample programs for Clustering and Association Rules.

The Discretization sample programs illustrate the techniques of binning, that is, forming groups of categorical values and ranges of numerical values to satisfy the requirements of ODM's algorithms.

The Model Seeker sample program illustrates the creation and testing of several supervised models with a variety of parameter settings; the model with the best test results is saved.

The Attribute Importance sample program illustrates the analysis of data in order to rank the variables by the influence of each in predicting target values.

The PMML sample programs illustrate the production and consumption (export/import) of data mining models conforming to the emerging standards for Predictive Model Markup Language.

The short sample programs `Sample_NaiveBayesBuild_short.java` and `Sample_NaiveBayesApply_short.java` illustrate basic ODM usage. They are described in detail in Chapter 3.

A.1.1 ODM Java API

This appendix does not include a detailed description of the ODM API classes and methods. For detailed information about the ODM API, see the ODM Javadoc in the directory `$ORACLE_HOME/dm/doc` (UNIX) or `%ORACLE_HOME%\dm\doc` (Windows) on any system where ODM is installed.

A.1.2 Oracle9i JDeveloper Project for the Sample Programs

If you want to use Oracle9*i* JDeveloper to exercise the sample programs, you can either create a new project or download an existing one. The ODM sample programs are available on Oracle Technology Network (`otn.oracle.com`) as an Oracle9*i* JDeveloper project. For information about the JDeveloper project, including installation instructions, see the readme file included with the download.

A.1.3 Requirements for Using the Sample Programs

The ODM user schema must be configured with or upgraded to Oracle9*i* release 2 (9.2.0.1).

The patch 9.2.0.2 must be applied.

The ODM user and ODM_MTR accounts must be unlocked if locked.

You will be required to provide the host name, TCP/IP port, and Oracle SID of the database to which you want to connect. Additionally, you will also need to know the password for ODM user on that database. Contact your database administrator if you do not know what those values are.

A.2 ODM Sample Programs Summary

Most programs have an (input) table. See Section A.4 for more information.

The sample programs, except for the short sample programs, use property files to specify values that control program execution. Each program has at least one property file. There is also one special property file, `Sample_Global.property`, that is used to specify the characteristics of the environment in which the programs run. See Section A.5 for more information.

After ODM is installed on your system, the sample programs, property files, and scripts are in the directory `$ORACLE_HOME/dm/demo/sample` (UNIX) or `%ORACLE_HOME%\dm\demo\sample` (Windows).

The rest of this section lists the ODM sample programs, arranged according to the ODM features that they illustrate. For detailed information about a program, see the comments in the sample program and in its property file.

A.2.1 Basic ODM Usage

The following sample programs are the programs that are discussed in detail in Chapter 3:

Note: If you execute `SampleNaiveBayesBuild.java` and then execute `SampleNaiveBayesBuild_short.java` or `Sample_NaiveBayesApply_short.java`, you must change `buildtablename` to a new name. Otherwise, you get a unique constraint error because the model name, MFS name, and Mining Task name are identical in both programs.

---

1. `Sample_NaiveBayesBuild_short.java`
   - Property file: This program does not have a property file.
   - Data: `census_2d_build_unbinned`

2. Sample_NaiveBayesApply_short.java
   - Property file: This program does not have a property file.
   - Data: census_2d_apply_unbinned

Neither of these sample programs uses either a property file or Sample_Global.property.

A.2.2 Adaptive Bayes Network Models

The following sample programs illustrate building an Adaptive Bayes Network Model, calculating lift for the model and testing it, and applying the model:

1. Sample_AdaptiveBayesNetworkBuild.java
   - Property file: Sample_AdaptiveBayesNetworkBuild.property
   - Data: census_2d_build_binned
2. Sample_AdaptiveBayesNetworkLiftAndTest.java
   - Property file: Sample_AdaptiveBayesNetworkLiftAndTest.property
   - Data: census_2d_test_binned
3. Sample_AdaptiveBayesNetworkApply.java
   - Property file: Sample_AdaptiveBayesNetworkApply.property
   - Data: census_2d_apply_binned

A.2.3 Naive Bayes Models

The following programs illustrate building a Naive Bayes Model, calculating lift for the model and testing it, applying the model, and cross validating the model:

Note: If you execute SampleNaiveBayesBuild.java and then execute SampleNaiveBayesBuild_short.java or Sample_NaiveBayesApply_short.java, you must change the buildtablename to a new name. Otherwise, you get a unique constraint error because the model name, MFS name, and Mining Task name are identical in both programs.

---

1. Sample_NaiveBayesBuild.java

- Property file: Sample_NaiveBayesBuild.property
- Data: census_2d_build_unbinned

2. Sample_NaiveBayesLiftAndTest.java
   - Property file: Sample_NaiveBayesLiftAndTest.property
   - Data: census_2d_test_unbinned
3. Sample_NaiveBayesApply.java
   - Property file: Sample_NaiveBayesApply.property
   - Data: census_2d_apply_unbinned
4. Sample_NaiveBayesCrossValidate.java
   - Property file: Sample_NaiveBayesCrossValidate.property
   - Data: census_2d_build_unbinned

A.2.4 Model Seeker Usage

The following sample program illustrates how to use Model Seeker to identify a "best" model:

1. Sample_ModelSeeker.java
   - Property file: Sample_ModelSeeker.property
   - Data: census_2d_build_unbinned and census_2d_test_unbinned

A.2.5 Clustering Models

The following sample programs illustrate building a clustering model, applying it, and inspecting clustering results:

1. Sample_ClusteringBuild.java
   - Property file: Sample_ClusteringBuild.property
   - Data: eight_clouds_build_unbinned
2. Sample_ClusteringApply.java
   - Property file: Sample_ClusteringApply.property
   - Data: eight_clouds_apply_unbinned
3. Sample_Clustering_Results.java
   - Property file: Sample_Clustering_Results.property.

- Input is name of built and applied clustering model

A.2.6 Association Rules Models

The following sample program illustrates building an Association Rules model:

`Sample_AssociationRules.java`

The property file depends on the format of the data:

- For transactional data:
  - Property file: `Sample_AssociationRules_Transactional.property`
  - Data: `market_basket_tx_binned`
- For nontransactional data:
  - Property file: `Sample_AssociationRules_TwoDimensional.property`
  - Data: `market_basket_2d_binned`

A.2.7 PMML Export and Import

The following sample programs illustrate importing and exporting PMML Models:

1. `Sample_PMML_Export.java`
   - Property file: `Sample_PMML_Export.property`
   - Data: no input data is required
2. `Sample_PMML_Import.java`
   - Property file: `Sample_PMML_Import.property`
   - Data: no input data is required

A.2.8 Attribute Importance Model Build and Use

The following sample programs illustrate how to build an attribute importance model and use the results to build another model:

1. `Sample_AttributeImportanceBuild.java`
   - Property file: `Sample_AttributeImportanceBuild.property`
   - Data: `magazine_2d_build_binned`

2. Sample_AttributeImportanceUsage.java
   - Property file: Sample_AttributeImportanceUsage.property
   - Data: magazine_2d_build_binned and magazine_2d_test_binned

A.2.9 Discretization

The following sample programs show to discretize (bin) data by creating a bin boundaries table and how to use the bin boundaries table:

1. Sample_Discretization_CreateBinBoundaryTables.java
   - Property file: Sample_Discretization_CreateBinBoundaryTables.property
   - Data: census_2d_build_unbinned
2. Sample_Discretization_UseBinBoundaryTables.java
   - Property file: Sample_Discretization_UseBinBoundaryTables.property
   - Data: census_2d_apply_unbinned

A.3 Using the ODM Sample Programs

After ODM is installed on your system, the sample programs, property files, and scripts are in the directory $ORACLE_HOME/dm/demo/sample (UNIX) or %ORACLE_HOME%\dm\demo\sample (Windows); the data used by the sample programs is in the directory $ORACLE_HOME/dm/demo/data (UNIX) or %ORACLE_HOME%\dm\demo\data (Windows). The data required by the sample programs is also installed in the ODM_MTR schema.

First, copy all of the sample files into a new directory so that the original files will remain intact.

Next, if necessary, connect to the database as the user ODM and run the script that starts the scheduler for ODM programs:

```
exec odm_start_monitor
```

The monitor must be started once in the life of a database installation; it is not harmful to start the monitor if it is already running. If a sample program is executed

and hangs at the beginning of a data mining task, then the monitor is probably not running.

Property files are used to specify common characteristics of the execution environment and to control execution of individual programs. `Sample_Global.property` file must be edited to point to the local database before any sample programs can be executed. The property files provide parameter settings for each of the sample programs; every parameter has a default setting, so the sample programs can be run without editing any of the property files. Each property file is discussed in Section A.5.

Scripts are included with the sample programs to compile and execute them. See Section A.6 for details.

The sample programs must be executed in the proper order. For a given model type, the sample build program must be executed before test, apply, or PMML export can be executed. For discretization, `CreateBinBoundaryTables` must be executed before `UseBinBoundaryTables`. The script that executes the sample programs supports a parameter that executes all of the programs in the correct order.

The sample programs illustrate the ODM API classes and methods used to perform various data mining tasks, and display the input required for each task as well as typical results.

Possible phases in exercising the sample code might be:

1. Compile and run a sequence of sample programs using all default values in the property files. To compile one or all of the sample programs, see Section A.6.1; to execute one of sample programs or all of them in order, see Section A.6.2.
2. For each program, make note of the input values, look at the source code to observe which ODM methods accomplish each piece of the process, and note the results.
3. Edit a property file to modify one or more parameters; re-execute the program and note changes in the results. Modifying the binning scheme requires editing the source code in `Sample_Discretization_CreateBinBoundaryTables.java`, which must then be re-compiled before execution.
4. Create new sample tables for building and testing a model from data that is not part of the supplied sample data. This will illustrate the data preparation that is required in order to implement a data mining solution within an existing application.

Note: The sample programs for Naive Bayes and Adaptive Bayes Network models require that any record to which you apply the models has either an integer or string attribute. You cannot apply the models to records that have a continuous numeric attribute.

A.4 Data Used by the Sample Programs

Each of the algorithms employed by ODM requires data that is discrete (binned) and numerical. Data for ODM programs can be binned in several ways:

- As a data preparation step performed before any ODM methods are executed
- Using ODM classes and methods as described in Section A.2.9
- Automatically by the ODM algorithms

For some of the sample programs, there is a choice of format for the input data, and binned as well as unbinned versions of the input data are supplied. The Discretization programs can be used to apply a customized binning scheme to unbinned data.

ODM can accept input data in either "nontransactional" format, that is one row per case, or "transactional" format, that is multiple rows per case. The input for the Association sample programs, the "Market Basket" data, is available in either format.

The data used to test a model must be distinct from the data that was used to build that model, but it is valid in a development setting to apply the model to the same data that was used for testing.

The data used by the sample programs is in the directory `$ORACLE_HOME/dm/demo/data` (UNIX) or `%ORACLE_HOME%\dm\demo\data` (Windows). The data is also installed in the `ODM_MTR` schema.

Table A–1 summarizes the data that is included with the sample programs.

Table A–1 ODM Sample Programs Data

Tables	Description
CENSUS_2D_BUILD_BINNED CENSUS_2D_BUILD_UNBINNED CENSUS_2D_TEST_BINNED CENSUS_2D_TEST_UNBINNED CENSUS_2D_APPLY_BINNED CENSUS_2D_APPLY_UNBINNED	The Census data is derived from information from the U.S. Census Bureau. The target attribute is CLASS, which represents salary level (0 = low salary and 1 = high salary). The Census data is used in all of the sample programs except in those illustrating clustering, attribute importance, and association rules.
EIGHT_CLOUDS_APPLY_UNBINNED EIGHT_CLOUDS_BUILD_UNBINNED	The Eight Clouds data is artificially generated for the sole purpose of illustrating Clustering.It is designed to produce eight partially overlapping clusters, which makes clustering this dataset nontrivial.
MAGAZINE_2D_BUILD_BINNED MAGAZINE_2D_TEST_BINNED	The Magazine data is derived from an actual marketing data set concerning a magazine subscription campaign. The target is MR_MAG, which represents Purchase (= 1) or No Purchase (= 0). This data is used in the Attribute Importance sample program.
MARKET_BASKET_2D_BINNED MARKET_BASKET_TX_BINNED	The Market Basket data represents shopping sessions in a grocery store. In the 2D (nontransactional) version, each row represents a shopping session and has a value 1 in the column for a product found in the check-out basket. This data is used in the Association Rules sample program.

A.5 Property Files for the ODM Sample Programs

After ODM is installed on your system, the sample programs, property files, and scripts are in the directory `$ORACLE_HOME/dm/demo/sample` (UNIX) or `%ORACLE_HOME%\dm\demo\sample` (Windows).

Most sample programs require two property files, `Sample_Global.property` and a property file specific to the sample program. The two short sample programs, `Sample_NaiveBayesBuild_short.java` and `Sample_NaiveBayesApply_short.java`, do not require any property files.

A.5.1 Sample_Global.property

The information in this file is used to make the connection to the database when a sample program is executed and to specify the location of the input and output tables.

A.5.1.1 Database and Schemas

During the installation of Oracle9*i*, the database name and port number were established. You must specify the URL for the database and the password for the ODM user in `miningServer.url`. The database URL is a string that specifies the type of JDBC driver used to connect to the database and database details. ODM supports the JDBC thin driver which requires a database URL in the following form:

"jdbc:oracle:thin:@<host_name>:<port_number>:<sid>"

The schemas (and user names) ODM and ODM_MTR were created during the installation, and during the Password Management phase, each schema user was assigned a password. Ensure that the password for ODM is entered in `miningServer.password`.

For an example of how to edit the global property file, see step 5 on page A-24.

The ODM schema is used internally by the ODM API programs; the ODM_MTR schema contains the sample input tables and some of the output tables. Normally you enter ODM_MTR for the values of both `inputDataSchemaName` and `outputSchemaName`.

A.5.1.2 Cleanup Section

Each sample program property file has a Cleanup Section. Since the sample programs, by default, re-use the names of objects created during execution, the default action is to delete the objects created during any previous execution of that program. You can choose to change the setting so that cleanup is prevented, but if you do, you must change all object names or risk program failure. You also have the option of cleaning up only, and not otherwise executing the program.

A.5.1.3 Tasks

Each distinct data mining operation is a task that is queued for execution. A task name is required to identify the operation, and each sample program has a default task name assigned in the parameter `miningTaskName`.

A.5.2 Sample_Discretization_CreateBinBoundaryTables.property

ODM algorithms require that input data be discretized (binned) before model building, testing, computing lift, and applying (scoring). You can either bin the data using appropriate Java methods or you can let the ODM algorithms automatically bin data. For a detailed discussion of discretization, see Section 1.8.

A.5.2.1 Sample Discretization Create Bin Boundary Input

If you use as input the default table `census_2d_build_unbinned`, then the `transactionalData` parameters are ignored. If you name a transactional table as `discretizationData.tableName`, then you must change `discretizationData.type` to `transactional` and enter the three column names of the table in the `transactionalData` parameters.

A.5.2.2 Sample Discretization Create Bin Boundary Output

This program creates separate binning definitions for categorical and numerical data and stores these definitions in the ODM_MTR schema in tables named in the parameters `discretization.discretizationNumericTableName` and `discretization.discretizationCategoricalTableName`.

A.5.3 Sample_Discretization_UseBinBoundaryTables.property

Section A.5.2 describes how to create bin boundaries. This section explains how to use those bin boundaries to create a bin boundaries table.

A.5.3.1 Sample Discretization Use Bin Boundary Input

If you use as input the default table `census_2d_apply_unbinned`, then the `transactionalData` parameters are ignored. If you name a transactional table as `discretizationData.tableName`, then you must change `discretizationData.type` to `transactional` and enter the three column names of the table in the `transactionalData` parameters.

The input table must have the same description as the table used in generating the Bin Boundary tables named in the input parameters `discretization.discretizationNumericTableName` and `discretization.discretizationCategoricalTableName`.

A.5.3.2 Sample Discretization Use Bin Boundary Output

The attribute values in the input data table are binned according to the rules in the Bin Boundary tables and the results are found in the view named in the parameter `discretization.discretizedViewName` in the ODM_MTR schema.

If the parameter `discretization.openEndedNumericalDiscretization` is set to `true`, then the highest and lowest bins are open-ended. That is, for example, instead of a top Age range of 90-100, the range will be “greater than 90”.

A.5.4 Sample_NaiveBayesBuild.property

If you use as input the default table census_2d_build_unbinned, then the `transactionalData` parameters are ignored. If you name a transactional table as `discretizationData.tableName`, then you must change `discretizationData.type` to `transactional` and enter the three column names of the table in the `transactionalData` parameters.

In addition, several parameters describe how the model is to be built; these settings are held in the object named in `classificationFunctionSettings.miningSettingsName`.

The setting `dataPrepStatus` indicates whether automatic binning will be used (`unprepared`) or whether the data has been previously binned (`discretized`).

The setting `supplementalAttributes` lists those attributes, in a comma-separated list, that could be used to identify individuals in a report, but cannot be used in building the model because they would eliminate the generality needed for the production of meaningful rules.

The setting `targetAttributeName` identifies the attribute value that the model will predict.

The predictions of the model are based on probabilities, which are based on the number of times particular values occur in the Build input data. Setting the Thresholds establishes how much data is gathered; in particular, raising the threshold eliminates "rarer" cases, making the rules tables smaller and the build process faster, but possibly making the model less accurate.

The parameter `naiveBayesOutput.modelName` assigns a name to the resultant model so that it can be uniquely identified in the testing and applying programs.

A.5.5 Sample_NaiveBayesLiftAndTest.property

There are two tasks included in Lift and Test.

- The Test task calculates the raw accuracy of the model when applied to data distinct from the data used to build the model (the test data is sometimes called the hold-out sample); it also calculates the "confusion matrix", which is a measure what type of errors are made by the model (false positive versus false negative).
- The Lift task measures how much better the model's predictions are than chance predictions. The lift calculations are reported on a quantile-by-quantile basis and the default number of quantiles is 10, that is, the results are reported on 10% subsets of the test data. One of the prediction values must be designated as the "positive" value. (This is easy if the predictions are "yes" versus "no", but even if the predictions are "red" versus "blue", one must be called "positive").

The parameter `liftAndTest.modelName` contains the name of the model produced by the `Sample_NaiveBayesBuild` program.

If you use as input the default table `census_2d_test_unbinned`, then the `transactionalData` parameters are ignored. If you name a transactional table as `discretizationData.tableName`, then you must change `discretizationData.type` to transactional and enter the three column names of the table in the `transactionalData` parameters.

The parameters for Lift are

- `numberOfQuantiles` (default = 10)
- `positiveTargetDisplayName` (the name used in reports, the default is `Positive Value`)
- `positiveTargetValue` (the actual value to be considered positive; default is 1),
- `positiveTargetDataType` (default is `int`; other possible values are: `char`, `float`, `boolean`, `string`).

The parameters c`omputeLift.resultName` and `test.resultName` give the names of the objects containing the test results.

A.5.6 Sample_NaiveBayesCrossValidate.property

The `Sample_NaiveBayesCrossValidate` program builds a Naive Bayes model and tests it by simulating an iterative process in which a model is built omitting one record of the input data, then the model is applied to that one record as a test. When

this procedure has been completed for each distinct record in the input data, an aggregate accuracy figure and confusion matrix are calculated. This process is an effective test when only a small number of records is available for model building.

The input parameters are the same as those for `Sample_NaiveBayesBuild` and `Sample_NaiveBayesLiftAndTest` and the results are the same as the results of `Sample_NaiveBayesLiftAndTest`, except that there are no Lift results.

A.5.7 Sample_NaiveBayesApply.property

The Naive Bayes model can be applied to new data to produce a scored list. The input data must be in the same format as the data used to build the model (attribute names and data types, binning scheme, dataset type – transactional or nontransactional). The input can be a table or a single record; in the case of a record, the column names must be in upper case and the model must have been built using automatic binning.

There are two output options for the format and contents of the resultant table: `multipleScoring` or `targetProbability`.

The choice of `multipleScoring` gives not only the predicted value and the probability (confidence) of the prediction, but also the option to list other possible target values and their probabilities. For example, if the target values are Low, Medium, High, then the output could be a single row with the prediction High and confidence.75, or three separate rows in the output table containing High / .75, Low / .15, Medium / .10.

The choice `targetProbability` gives a result for each target value, and in addition to the information produced by multipleScoring, displays the ranking of each predicted value.

`applyOutputOption` is `multipleScoring` or `targetProbability`, as explained above. The parameter `sourceAttributeNames` is a comma-separated list of the attribute names from the input table to be included in the output; `sourceAttributeAliases` give a display name for each attribute.

For both `multipleScoring` and `targetProbability`, `predictionColumnName` and `probabilityColumnName` give the names of the columns in the result table containing the predicted value and the confidence. The parameter `numberOfPredictions` is an integer from 1 to the number of distinct target values, and indicates how many rows will be produced for each input record. The parameter `topToBottom` is `true` if you want the predictions sorted in descending order by probability and `false` if you want the predictions sorted in ascending order.

The parameter `rankAttribute` is the column name for rank in the result table.

The distinct possible prediction values are listed in `targetValues` (with data type `targetDataType`), and the display names for the predictions are listed in `targetDisplayNames`.

A.5.8 Sample_AttributeImportanceBuild.property

The parameter `dataPrepStatus` indicates whether automatic binning should be applied to the input data (`unprepared`) or if the data is already binned (`discretized`).

The target attribute is named in `targetAttributeName`.

The resultant table contains a list of the attributes ranked by their influence in predicting the target value.

A.5.9 Sample_AttributeImportanceUsage.property

This program executes the Attribute Importance program on the input data, then uses one of five methods available to create a table containing a subset of the original attributes. It then builds two Naive Bayes models, one using the full attribute set (NB1), the other using the reduced set as input (NB2), and displays the two test results.

The threshold and target parameters are set as for Naive Bayes Build. The accuracy parameter is ignored in the sample code.

The attributeSelection parameter has an integer value from 1 to 5, as follows:

1. Specify `1` to select attributes whose Importance is above or below (inclusive) a given value.

   Example: attributes with importance value above 0.01:

   ```
   threshold = 0.01, aboveThreshold = true
   ```

2. Specify `2` to select attributes whose Importance is between two given values

   Example: attributes with importance value between 0.01 and 0.02:

   ```
   lowerBound = 0.01, upperBound = 0.02
   ```

3. Specify `3` to select attributes by rank from a list sorted in descending order by Importance, above or below a given rank

   Example: the first five entries from the list:

```
threshold = 5, aboveThreshold = true
```

4. Specify 4 to select attributes by rank from a list sorted in descending order by Importance, between two given ranks

   Example: attributes with rank between 3 and 6:

   ```
   lowerBound = 3, upperBound = 6
   ```

5. Specify 5 to select the highest N% or lowest N% of attributes from a list sorted in descending order by Importance

   Example: the first 10% of attributes from the list

   ```
   percentage = 0.1, aboveThreshold = true
   ```

A.5.10 Sample_AssociationRules Property Files

The Sample_AssociationRules program takes one of two property files depending on the format of the input data table

- `Sample_AssociationRules_TwoDimensional.property` (Input table is nontransactional.)
- `Sample_AssociationRules_Transactional.property` (Input table is transactional.)

> **Note:** Since there is a choice of property files**,** unlike the other sample programs, the name of the property file must be explicitly specified.

The `buildData.type` parameter is set appropriately to `transactional` or `nonTransactional`, depending on which property file is used, and in the transactional case, the column headings must be set. Otherwise the two property files have the same parameters.

The function settings parameters are as follows:

- `dataPrepStatus` is set to `discretized` (binned) or `unprepared` (unbinned); the sample input tables MARKET_BASKET_TX_BINNED and MARKET_BASKET_2D_BINNED are already binned.
- `minimumSupport` is a value between 0 and 1 that specifies a threshold for the frequency of the components of a rule occurring together. For example, if the rule is “A implies B”, then the number of cases in which A and B occur

simultaneously are counted. Itemsets below the minimum are eliminated, making the calculations faster.

- `minimumConfidence` specifies a threshold for the conditional probability for a rule. Rules with probability below the threshold are eliminated, making the calculations faster.
- `maximumRuleLength` specifies the number of components in a rule. The default setting in the sample property files is 2, forcing all rules to be of the type "A implies B", rather than "A and B imply C", or longer.

A.5.11 Sample_ModelSeeker.property

The Model Seeker sample program creates several Naive Bayes (NB) and Adaptive Bayes Network (ABN) models and compares them by calculating a value called the "Figure of Merit" based on the test results of the models. The model with the best Figure of Merit is saved; for other models, the models are discarded and only the settings and test results are saved.

Build and Test input data tables are identified, with parameter settings as for the Naive Bayes sample programs.

The `AlgorithmSetting` parameters specify the model types to be built and the input parameters for each. The `Setting_1` specifies one NB model and `Setting_4` one ABN model; the parameters are as described for the property files of those model types. Settings 2, 3, and 5 specify multiple models and a list of values is entered for some parameters. For NB, the setting `crossProduct` indicates that models with every possible combination of parameter values are built, whereas `parallelPairs` indicates that a model is built using the first value in each list, another model is built using the second value in each list, and so on.

The function settings include `dataPreparationStatus` (`discretized` = binned, `unprepared` = unbinned), `targetAttributeName`, `targetAttributeType` (`categorical`, `numerical`), and `supplementalAttributes` (a comma-separated list of attributes to be ignored during model build).

For the purposes of calculating Lift, one target value is designated as “positive”. The value, data type, and Name used in reports are established in the `positiveCategoryTarget` parameters.

The parameter `modelSeeker.weight` assigns a value used in calculating the Figure of Merit. The default value of 1 makes no distinction in cost between types of errors in prediction: false-positive (a prediction of positive when the actual value is negative) and false-negative (a prediction of negative when the actual value is

positive). An example is a model to predict that a customer is likely to buy Product X in a telemarketing campaign: the cost of a false-positive is small (the cost of a telephone call) but the cost of a false-negative is high (the revenue for a lost sale). In this case a weight greater than 1 would be set (by experiment). In a case in which the cost of false-positive is higher than false-negative, a weight between 0 and 1 would be used.

The parameter `modelSeeker.liftQuantiles` sets the number of quantiles displayed in the Lift table. For large Test data tables, more finely-grained results can be observed for lift quantiles set to 100.

A.5.12 Sample_ClusteringBuild.property

There are two clustering algorithms available in ODM:

- A version of *k*-means enhanced to provide hierarchies in the clustering
- A proprietary algorithm called O-cluster.

For *k*-means, you define the (maximum) number of clusters, whereas O-cluster determines the number of clusters as part of the algorithm. The default input table is an artificially-generated set of data designed to illustrate the functionality by forming eight distinct clusters.

The clustering parameters include:

- `algorithmType` determines the algorithm to be used: 1 = *k*-means, 2 = O-cluster.
- `clusters` sets a maximum number of clusters.
- For O-cluster, the `sensitivity` parameter is a value between 0 and 1 that measures the sharpness of the cluster definitions: a lower sensitivity normally produces fewer clusters.
- For *k*-means, the `error`, `iterations`, and `stoppingCriterion` parameters determine how long the build process will take. The `error` parameter measures the amount of change resulting from one more iteration; a lower error value normally allows more iterations. The `iterations` parameter is a maximum value and the `stoppingCriterion` parameter indicates whether to rely only on error, only on iterations, or on a combination of the two (whichever occurs first) to determine when to stop the process.
- The `dataPrepStatus` indicates whether the input data is binned (`discretized`) or unbinned (`unprepared`). If unbinned, automatic binning will occur.

A.5.13 Sample_ClusteringApply.property

The Clustering Apply program takes the rules generated by Clustering Build and applies them to a single record or to a table to identify the cluster membership of each record. If the input is a single record, then the clustering model must have been built using automatic binning.

The parameter `apply.type` indicates whether the input is a record or a table. The `modelName` is the `clusteringOutput.modelName` from the Clustering Build program. The `miningApplyOutput.ApplyOutputOption` indicates the format of the results table, as follows:

- `multipleScoring`: for each record, one row is created for each cluster (up to the number of clusters specified in `numberOfPredictions`) containing columns giving the cluster number (`targetAttribute`) and probability that the given record is a member of that cluster (`probabilityTargetAttribute`), in descending order of probability (`topToBottom` = `true`) or ascending (`topToBottom` = `false`).
- `targetProbability`: produces the same format as for multipleScoring except that scores are given only for the clusters listed in the parameter `targetValues` (with display names in `targetDisplayName`), and in addition the ranking of each cluster for the given record is stored in the column named in the parameter rankAttribute.
- The parameter `sourceAttributeNames` is a comma-separated list of the attributes from the input table to be included in the output, with display names in `sourceAttributeAliases`.

A.5.14 Sample_Clustering_Results.property

`Sample_Clustering_Results.java` illustrates how to get information about the results of applying a clustering model. `Sample_Clustering_Results.java` returns the following kinds of information:

- General information: number of clusters, number of records, number of levels in the hierarchy.
- Information about each cluster: record count and attribute values for the centroid
- Other information: rules for each cluster and a sample histogram for one of the clusters

`Sample_Clustering_Results.property` requires on input value: the name of the clustering model built with `Sample_ClusteringBuild.java` and applied using `Sample_ClusteringApply.java`.

A.5.15 Sample_AdaptiveBayesNetworkBuild.property

An Adaptive Bayes Network can be thought of as a series of small trees that can be combined to produce a set of rules describing how the model makes predictions. These small trees are called Network Features. Each level of a network feature describes one attribute, and the number of branches at each node is the number of distinct values of that attribute. Thus, features can quickly become complex; there are several parameters in Adaptive Bayes Network that control the complexity and therefore the build time:

- `maximumNetworkFeatureDepth`: the maximum number of levels in each feature.The default is 10.
- `maximumNumberOfNetworkFeatures`: the features are ranked using a calculation called MDL (Minimum Description Length), and the top N, as specified by this parameter, are used. The default is 10.
- `maximumConsecutivePrunedNetworkFeatures`: after one feature is built, a second feature is built and a test based on MDL determines whether the two features together have more predictive power than feature 1 alone. If not, feature 2 is eliminated, a new feature 2 is built and the test is repeated. If N new features are built and eliminated, then feature 1 alone is selected, where N is the value of this parameter. In addition, a Naive Bayes model is built and tested against the "winning" ABN model.
- `maximumBuildTime`: sets a maximum time (in minutes) for the model build
- `priorProbabilities`: if the distribution of target values in the build data set is very different from the distribution in a random sample of data, then the actual distribution is described in these parameters in order to produce realistic predictions. The parameter targetValues is a comma-separated list of values in the target attribute, and `targetProbabilities` lists the percentage of records having each of the values.
- `categoryMatrix`: these parameters describe the cost matrix that will cause bias in the predictions towards one particular value by specifying the cost of each type of mistake. The matrix in the comments of the property file illustrates that a false negative prediction (predict 0 where the actual value is 1) is twice as costly as a false positive prediction. The list of values in `costValues` will fill

the rows of the matrix in order. (see also the discussion of weights for the Model Seeker Build program)

Other parameters have the same meaning as for Naive Bayes as described in Section A.5.4.

A.5.16 Other Sample_AdaptiveBayesNetwork Property Files

The parameters in the property files `SampleAdaptiveBayesNetworkLiftAndTest.property` and `SampleAdaptiveBayesNetworkApply.property` are identical to those of the corresponding Naive Bayes property file:

- `Sample_NaiveBayesLiftAndTest.property` (Section A.5.5)
- `Sample_NaiveBayesApply.property` (Section A.5.7)

A.5.17 Sample PMML Import and Export Property

ODM provides programs to export a model into a table and import a model from a table in a format conforming to emerging standards called PMML. These standards will allow a model created by one vendor's data mining utility to be used by another vendor's utility for scoring.

The parameters specify the model to be imported/exported as well as the schema, table, and column containing the model specifications.

A.6 Compiling and Executing ODM Sample Programs

This section provides a brief description of how to compile and execute the ODM sample programs. You can do the following:

- Compile one or all of the sample programs
- Execute a sample program or all sample programs in the correct order

After ODM is installed on your system, the sample programs, property files, and scripts are in the directory `$ORACLE_HOME/dm/demo/sample` (UNIX) or `%ORACLE_HOME%\dm\demo\sample` (Windows).

The sample programs first check to see if they have run previously. If the program has run previously, each cleans up the environment before it runs again. You can execute these programs more than once without getting errors.

A.6.1 Compiling the Sample Programs

Follow these steps to compile the sample programs:

1. Set your ORACLE_HOME environment variable.
2. Ensure that you have installed JDK 1.3.1 or above. JDK 1.3.1 may also be available in ORACLE_HOME with your installation. Set your JAVA_HOME environment variable; it should point to your installed JDK directory or the one available in ORACLE_HOME.
3. On UNIX, you must set your CLASSPATH environment variable so that it includes the following Oracle9i Java Archive files:

   ```
   $ORACLE_HOME/jdbc/lib/classes12.jar
   $ORACLE_HOME/lib/xmlparserv2.jar
   $ORACLE_HOME/rdbms/jlib/jmscommon.jar
   $ORACLE_HOME/rdbms/jlib/aqapi.jar
   $ORACLE_HOME/rdbms/jlib/xsu12.jar
   $ORACLE_HOME/dm/lib/odmapi.jar
   ```

   On Windows, your Classpath system variable must include

   ```
   %ORACLE_HOME%\jdbc\lib\classes12.jar
   %ORACLE_HOME%\lib\xmlparserv2.jar
   %ORACLE_HOME/rdbms/jlib/jmscommon.jar
   %ORACLE_HOME%\rdbms\jlib\aqapi.jar
   %ORACLE_HOME%\rdbms\jlib\xsu12.jar
   %ORACLE_HOME\dm\lib\odmapi.jar
   ```

   If you use a database character set that is not US7ASCII, WE8DEC, WE8ISO8859P1, or UTF8, you must also include the following in your CLASSPATH:

   `$ORACLE_HOME/jdbc/lib/nls_charset12.zip` (on UNIX)
   `%ORACLE_HOME%\jdbc\lib\nls_charset12.zip` (on Windows)

4. Before you compile the short programs `Sample_NaiveBayesBuild_short.java` and `Sample_NaiveBayesApply_short.java`, you must edit the programs to specify the database URL (DB_URL) and the password for the ODM user. The database URL is a string that specifies the type of JDBC driver used and database details. ODM supports the JDBC thin driver which requires a database URL in the following form:

   ```
   "jdbc:oracle:thin:@<host_name>:<port_number>:<sid>"
   ```

To specify the data mining server, substitute appropriate values for DB_URL and ODM password in the following line in both short sample programs:

```
dms = new DataMiningServer("DB_URL", "odm", "ODM password");
```

The location access data has already been specified in both programs for your convenience.

5. For all other sample programs except for the short sample programs, you must edit `Sample_Global.property` file to specify the database URL, the ODM user name, and the password for ODM user. Replace the strings `MyHost`, `MyPort`, `MySid`, `MyName`, and `MyPW` with the appropriate values for your system. `MyName` refers to the ODM user and it must be replaced with `odm`. `MyPW` is the password specified during ODM configuration or while unlocking the ODM user account. By default the password is `odm`.

   For example:

   ```
   miningServer.url=jdbc:oracle:thin:@odmserver.company.com:1521:orcl
   miningServer.userName=odm
   miningServer.password=odm
   inputDataSchemaName=odm_mtr
   outputSchemaName=odm_mtr
   ```

6. You can compile each ODM sample program or all of them at once by running one of the provided scripts.

   To compile a specific sample program, execute one of the following scripts as shown:

   On UNIX platforms:

   ```
   /usr/bin/sh compileSampleCode.sh <filename>
   ```

   For example, to compile SampleModelSeeker.java:

   ```
   /usr/bin/sh compileSampleCode.sh Sample_ModelSeeker.java
   ```

   On Windows platforms:

   ```
   compileSampleCode.bat <filename>
   ```

   For example, to compile SampleModelSeeker.java:

   ```
   compileSampleCode.bat Sample_ModelSeeker.java
   ```

To compile all of the programs, use one of the scripts with the parameter `all`:

On UNIX platforms:

```
/usr/bin/sh compileSampleCode.sh all
```

On Windows platforms:

```
compileSampleCode.bat all
```

A.6.2 Executing the Sample Programs

1. Before you execute a sample program, log in to the ODM user schema in the database and type the following command to turn on the ODM task monitor

   ```
   exec odm_start_monitor
   ```

   Generally, you will not have to start the monitor again unless you manually stop it or the job associated with the task monitor turns broken. If you do not start the task monitor, any data mining tasks pending for execution will hang.

2. Each of the sample programs uses `Sample_Global.property` and a program-specific property file. For example, `Sample_ModelSeeker.java` requires `Sample_ModelSeeker.property`. To execute a specific sample program use one of the provided scripts. If you do not specify a property file, the script assumes the default property file for the specified program. Note that `Sample_AssociationRules.java` has a choice of two distinct property files; the desired property file for this sample program must be specified explicitly.

   The short sample programs do not require property files. To execute them, use the script without specifying any property files.

   To execute a specific sample program, execute one of the scripts as follows:

   On UNIX platforms:

   ```
   /usr/bin/sh executeSampleCode.sh <classname> [<property file>]
   ```

   For example:

   ```
   /usr/bin/sh executeSampleCode.sh Sample_ModelSeeker
   /usr/bin/sh executeSampleCode.sh Sample_ModelSeeker myFile.property
   ```

   On Windows platforms:

   ```
   executeSampleCode.bat <classname> [<property file>]
   ```

For example:

```
executeSampleCode.bat Sample_ModelSeeker
executeSampleCode.bat Sample_ModelSeeker myFile.property
```

3. The ODM sample programs must be executed in the correct order; for example,:
   - For a given model type, the sample build program must be executed before test, apply, or PMML export can be executed.
   - For discretization, `Sample_Discretization_CreateBinBoundaryTables` must be executed before `Sample_Discretization_UseBinBoundaryTables` can be executed.
   - You must execute `Sample_NaiveBayesBuild_short` before `Sample_NaiveBayesApply_short` can be executed.

   To execute all of the programs in the correct order (except for `Sample_NaiveBayesBuild_short.java` and `Sample_NaiveBayesApply_short.java`), use one of the execution scripts with the parameter `all`:

   On UNIX platforms:

   ```
   /usr/bin/sh executeSampleCode.sh all
   ```

   On Windows platforms:

   ```
   executeSampleCode.bat all
   ```

Note: If you use the `all` parameter, you must use the default names for the program-specific property files. Association rules models are executed twice using both of the possible property files described in Section A.5.10.

Glossary

algorithm

A specific technique or procedure for producing a data mining model. An algorithm uses a specific model representation and may support one or more functional areas. Examples of algorithms used by ODM include Naive Bayes and Adaptive Bayes Networks for classification, *k*-means and O-Cluster for clustering, predictive variance for attribute importance, and Apriori for association rules.

algorithm settings

The settings that specify algorithm-specific behavior for model building.

apply output

A user specification describing the kind of output desired from applying a model to data. This output may include predicted values, associated probabilities, key values, and other supplementary data.

association rules

A data mining function that captures co-occurrence of items among transactions. A typical rule is an implication of the form A -> B, which means that the presence of itemset A implies the presence of itemset B with certain support and confidence. The support of the rule is the ratio of the number of transactions where the itemsets A and B are present to the total number of transactions. The confidence of the rule is the ratio of the number of transactions where the itemsets A and B are present to the number of transactions where itemset A is present. ODM uses the Apriori algorithm for association rules.

attribute

An instance of `Attribute` maps to a column with a name and data type. The attribute corresponds to a column in a database table. When assigned to a column, the column must have a compatible data type; if the data type is not compatible, a runtime exception is likely. Attributes are also called *variables, features, data fields*, or *table columns*.

attribute importance

A measure of the importance of an attribute in predicting a specified target. The measure of different attributes of a build data table enables users to select the attributes that are found to be most relevant to a mining model. A smaller set of attributes results in a faster model build; the resulting model could be more accurate. ODM uses the predictive variance algorithm for attribute importance. Also known as *feature selection* and *key fields*.

attribute usage

Specifies how a logical attribute is to be used when building a model, for example, active or supplementary, suppressing automatic data preprocessing, and assigning a weight to a particular attribute. See also **attributes usage set**.

attributes usage set

A collection of attribute usage objects that together determine how the logical attributes specified in a logical data object are to be used.

binning

See **discretization**.

case

All the data collected about a specific transaction or related set of values.

categorical attribute

An attribute where the values correspond to discrete categories. For example, *state* is a categorical attribute with discrete values (CA, NY, MA, etc.). Categorical attributes are either non-ordered (nominal) like state, gender, etc., or ordered (ordinal) such as high, medium, or low temperatures.

category

Corresponds to a distinct value of a categorical attribute. Categories may have string or numeric values. String values must not exceed 64 characters in length.

centroid

See **cluster centroid**.

classification

A data mining function for predicting target values for new records using a model built from records with known target values. ODM supports two algorithms for classification, Naive Bayes and Adaptive Bayes Networks.

cluster centroid

The cluster centroid is the vector that encodes, for each attribute, either the mean (if the attribute is numerical) or the mode (if the attribute is categorical) of the cases in the build data assigned to a cluster.

clustering

A data mining function for finding naturally occurring groupings in data. More precisely, given a set of data points, each having a set of attributes, and a similarity measure among them, clustering is the process of grouping the data points into different clusters such that data points in the same cluster are more similar to one another and data points in different clusters are less similar to one another. ODM supports two algorithms for clustering, *k*-means and O-Cluster.

confusion matrix

Measures the correctness of predictions made by a model from a text task. The row indexes of a confusion matrix correspond to *actual values* observed and provided in the test data. These were used for model building. The column indexes correspond to *predicted values* produced by applying the model. For any pair of actual / predicted indexes, the value indicates the number of records classified in that pairing.

When predicted value equals actual value, the model produces correct predictions. All other entries indicate errors.

cost matrix

A two-dimensional, *n* by *n* table that defines the cost associated with a prediction versus the actual value. A cost matrix is typically used in classification models, where *n* is the number of distinct values in the target, and the columns and rows are labeled with target values. The rows are the actual values; the columns are the predicted values.

cross-validation

A technique of evaluating the accuracy of a classification or regression model. This technique is used when there are insufficient cases for model building and testing. The data table is divided into several parts, with each part in turn being used to evaluate a model built using the remaining parts. Cross-validation occurs automatically for Naive Bayes and Adaptive Bayes networks.

data mining

The process of discovering hidden, previously unknown, and usable information from a large amount of data. This information is represented in a compact form, often referred to as a *model*.

data mining server (DMS)

The component of the Oracle database that implements the data mining engine and persistent metadata repository.

discretization

Discretization groups related values together under a single value (or bin). This reduces the number of distinct values in a column. Fewer bins result in models that build faster. ODM algorithms require that input data be *discretized* prior to model building, testing, computing lift, and applying (scoring).

distance-based (clustering algorithm)

Distance-based algorithms rely on a distance metric (function) to measure the similarity between data points. Data points are assigned to the nearest cluster according to the distance metric used.

DMS

See **data mining server (DMS)**.

feature

See **network feature**.

lift

A measure of how much better prediction results are using a model than could be obtained by chance. For example, suppose that 2% of the customers mailed a catalog without using the model would make a purchase. However, using the model to select catalog recipients, 10% would make a purchase. Then the lift is 10/2 or 5. Lift may also be used as a measure to compare different data mining models. Since lift is computed using a data table with actual outcomes, lift compares how well a model performs with respect to this data on predicted outcomes. Lift indicates how well the model improved the predictions over a random selection given actual results. Lift allows a user to infer how a model will perform on new data.

location access data

Specifies the location of data for a mining operation.

logical attribute

A description of a domain of data used as input to mining operations. Logical attributes may be categorical, ordinal, or numerical.

logical data

A set of mining attributes used as input to building a mining model.

MDL principle

See **minimum description length principle**.

minimum description length principle

Given a sample of data and an effective enumeration of the appropriate alternative theories to explain the data, the best theory is the one that minimizes the sum of

- The length, in bits, of the description of the theory
- The length, in bits, of the data when encoded with the help of the theory

mining apply output

See **apply output**.

mining function

ODM supports the following mining functions: classification, association rules, attribute importance, and clustering.

mining function settings

An object that specifies the type of model to build, the function of the model, and the algorithm to use. ODM supports the following mining functions: classification, association rules, attribute importance, and clustering.

mining model

The result of building a model from mining function settings. The representation of the model is specific to the algorithm specified by the user or selected by the DMS. A model can be used for direct inspection, e.g., to examine the rules produced from a decision tree or association rules, or to score data.

mining result

The end product(s) of a mining operation. For example, a build task produces a mining model; a test task produces a test result.

missing value

A data value that is missing because it was not measured (that is, has a null value), not answered, was unknown, or was lost. Data mining systems vary in the way they treat missing values. Typically, they ignore missing values, omit any records containing missing values, replace missing values with the mode or mean, or infer

missing values from existing values. ODM ignores missing values during mining operations.

mixture model

A mixture model is a type of density model that includes several component functions (usually Gaussian) that are combined to provide a multimodal density.

model

An important function of data mining is the production of a model. A model can be descriptive or predictive. A descriptive model helps in understanding underlying processes or behavior. For example, an association model describes consumer behavior. A predictive model is an equation or set of rules that makes it possible to predict an unseen or unmeasured value (the dependent variable or output) from other, known values (independent variables or input). The form of the equation or rules is suggested by mining data collected from the process under study. Some training or estimation technique is used to estimate the parameters of the equation or rules. See also **mining model**.

multi-record case

See **transactional format**.

network feature

A network feature is a tree-like multi-attribute structure. From the standpoint of the network, features are conditionally independent components. Features contain at least one attribute (the root attribute). Conditional probabilities are computed for each value of the root predictor. A two-attribute feature will have, in addition to the root predictor conditional probabilities, computed conditional probabilities for each combination of values of the root and the depth 2 predictor. That is, if a root predictor, *x*, has *i* values and the depth 2 predictor, *y*, has *j* values, a conditional probability is computed for each combination of values {*x=a*, *y=b* such that *a* is in the set {1,..,*i*} and *b* is in the set {1,..,*j*}}. Similarly, a depth 3 predictor, *z*, would have additional associated conditional probability computed for each combination of values {*x=a*, *y=b*, z=*c* such that *a* is in the set {1,..,*i*} and *b* is in the set {1,..,*j*} and *c* is in the set {1,..,*k*}}. Network features are used in the Adaptive Bayes Network algorithm.

nontransactional format

Each case in the data is stored as one record (row) in a table. Also known as *single-record case*. See also **transactional format**.

numerical attribute

An attribute whose values are numbers. The numeric value can be either an integer or a real number. Numerical attribute values can be manipulated as continuous values. See also **categorical attribute**.

outlier

A data value that does not (or is not thought to have) come from the typical population of data; in other words, a data value that falls outside the boundaries that enclose most other data values in the data.

physical data

Identifies data to be used as input to data mining. Through the use of attribute assignment, attributes of the physical data are mapped to logical attributes of a model's logical data. The data referenced by a *physical data* object can be used in model building, model application (scoring), lift computation, statistical analysis, etc.

physical data specification

An object that specifies the characteristics of the physical data used in a mining operation. The physical data specification includes information about the format of the data (transactional or nontransactional) and the roles that the data columns play.

positive target value

In binary classification problems, you may designate one of the two classes (target values) as positive, the other as negative. When ODM computes a model's lift, it calculates the density of positive target values among a set of test instances for which the model predicts positive values with a given degree of confidence.

predictor

A logical attribute used as input to a supervised model or algorithm to build a model.

prior probabilities

The set of prior probabilities specifies the distribution of examples of the various classes in data. Also referred to as *priors*, these could be different from the distribution observed in the data.

priors

See **prior probabilities**.

rule

An expression of the general form *if X, then Y*. An output of certain models, such as association rules models or decision tree models. The predicate *X* may be a compound predicate.

score

Scoring data means applying a data mining model to new data to generate predictions. See **apply output**.

settings

See **algorithm settings** and **mining function settings**.

single-record case

See **nontransactional format**.

supervised mining (learning)

The process of building data mining models using a known dependent variable, also referred to as the target. Classification techniques are supervised. See **unsupervised mining (learning)**.

target

In supervised learning, the identified logical attribute that is to be predicted. Sometimes called *target value* or *target attribute*.

task

A container within which to specify arguments to data mining operations to be performed by the data mining system.

transactional format

Each case in the data is stored as multiple records in a table with columns `sequenceID`, `attribute_name`, and `value`. Also known as *multi-record case*. See also **nontransactional format**.

transformation

A function applied to data resulting in a new form or representation of the data. For example, discretization and normalization are transformations on data.

unsupervised mining (learning)

The process of building data mining models without the guidance (supervision) of a known, correct result. Clustering and association rules are unsupervised mining functions. See **supervised mining (learning)**.

Index

A

B

C

M

N

O

P

R

S

T

U