Oracle9*i* Data Mining

Concepts

Release 2 (9.2)

March 2002

Part No. A95961-01

ORACLE

Oracle9*i* Data Mining Concepts, Release 2 (9.2)

Part No. A95961-01

Contents

Send Us Your Comments

Oracle9*i* Data Mining Concepts, Release 2 (9.2)

Part No. A95961-01

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

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

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

- DARWINDOC@us.oracle.com
- 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 manual describes how to use the Oracle9*i* 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 scoring new data using that model.
- 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 *Oracle9*i *Database Documentation Library*. The ODM documentation set consists of the following documents, available online:

- *Oracle9*i *Data Mining Administrator's Guide*, Release 2 (9.2)
- *Oracle9*i *Data Mining Concepts*, Release 2 (9.2) (this document)

For last minute information about ODM, see the *Oracle9*i *README*, Release 2 (9.2) 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:

- *Oracle9*i *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. This enables Oracle9*i* to provide an infrastructure for 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 that provides complete programmatic control of data mining functions to deliver data mining within the database.

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 is enhanced with several data mining capabilities: decision trees, clustering, attribute importance (also known as feature selection), and others, as described below.

- **Decision Trees (Adaptive Bayes Network):** Expands ODM support of supervised learning techniques (learning techniques that predict a target value). Functionality is implemented using Adaptive Bayes Network technology. ODM can be used to make predictions with an associated probability.

  A significant benefit of decision trees is that they produce a set of 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 probablistic cluster membership assignment and cluster rules.

- **Attribute Importance:** Can be 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 referred to as *feature selection* or *key fields*.

- **Model Seeker:** Automatically builds multiple data mining models with minimal user input, compares the models, 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 boundaries and later modifies them to override the created bins. With automatic preprocessing, there is no modifying the boundaries after they are set. See Section 1.7.2, "Discretization".

- **Predictive Model Markup Language (PMML):** 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 1.8, "PMML Support".

- **Mining Task:** All data mining operations (build, test, compute lift, apply, import, and export) are performed asynchronously using a mining task. The static methods supported by ODM 9.0.1 for these mining operations are deprecated in this release. Mining tasks further allow obtaining status on the execution of mining operations.

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's Java Community Process as a Java Specification Request (JSR-73). JDM is based on 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. 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 its 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 in each case. All but one attribute is a predictor attribute, and one is the target attribute. 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 or not 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.

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 such as trees. 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. With clustering models, the data density itself drives the process to a final solution, that is, determine clusters.

If the components overlap (mix), we have soft assignment of data points to clusters. If the clusters are organized into a hierarchical structure, clustering implicitly defines a taxonomy for the data.

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 function 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.[1]

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

[1] Association rule mining can be formally defined as follows: Let $I = \{i_1, i_2, \ldots, i_n\}$ 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 => Y (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.

"frequent item sets") are discovered. In the second phase, rules of the form X => 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 = > B) = P(A, B).
- **Confidence:** Confidence of a rule is the conditional probability of B given A; confidence (A => B) = P (B | 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 ones 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 supporting decision trees (classification)
- Naive Bayes (classification)
- Model Seeker (classification)
- *k*-Means (clustering)
- O-Cluster (clustering)
- Predictive 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

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 trees. ABN provides a fast, scalable, non-parametric[2] means of extracting predictive information from data with respect to a target attribute. 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, marketer, 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 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. 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 uses costs and priors for both building and scoring (see Section 1.3.1.1, "Costs" and Section 1.3.1.2, "Priors").

1.4.1.1 Max Build Parameters

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

- **MaximumNetworkFeatureDepth**: Network features[3] are like individual decision trees. This parameter restricts the depth of any individual *NetworkFeature* in the model. At each depth for an individual *NetworkFeature*, there is only one predictor. Computation cost goes up by depth with the product of the number of unique values of the predictors of depth less than or equal to the depth, so the computational cost of deep feature builds is 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. The default is 10.

[2] 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.

- **Maximum Number of Network Features:** Controls the maximum number of features included in this ABN 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 provide a statistical boost 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.
- **Maximum Consecutive Prune Network Features:** The maximum number of consecutive pruned features before halting the stepwise selection process. Negative values are used to indicate that only the Naive Bayes model and a single feature 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 5.
- **Maximum Build Time:** 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

[3] 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 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*]}.

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.
- **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 1–2, 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 accuracy estimate is a type of cross-validation called "leave-one-out cross-validation".[4]

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.To use Naive Bayes cross-validation, the user executes a `MiningTaskCrossValidate` task, specifying that a Naive Bayes model is to be

[4] "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.

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 rows	Any size	Any size
Number of attributes	Any number (built-in feature selection)	Best if less than 200
Speed	Not as fast	Faster
Accuracy	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 asynchronously build multiple classifications and evaluate the models and select a "best" model.

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.

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.

1.4.4.2 Summarization

Because *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 generates more points for regions of the input space where it is harder to separate clusters.

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 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. 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 unhandiest *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. 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. 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 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 rows	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 less than 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.

In particular, for each attribute *n*, for each possible value *i*, and for each possible value *k* of the target attribute, we tabulate

P(attribute-has-value-*i* | target-attribute-has-value-*k*)

where P means "probability of" and "|" means "given that". These statistics give an idea of how each attribute correlates with the target attribute. The higher the correlation of an attribute with the target attribute, the more useful ODM's Attribute Importance ranks it.

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 -> 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 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.

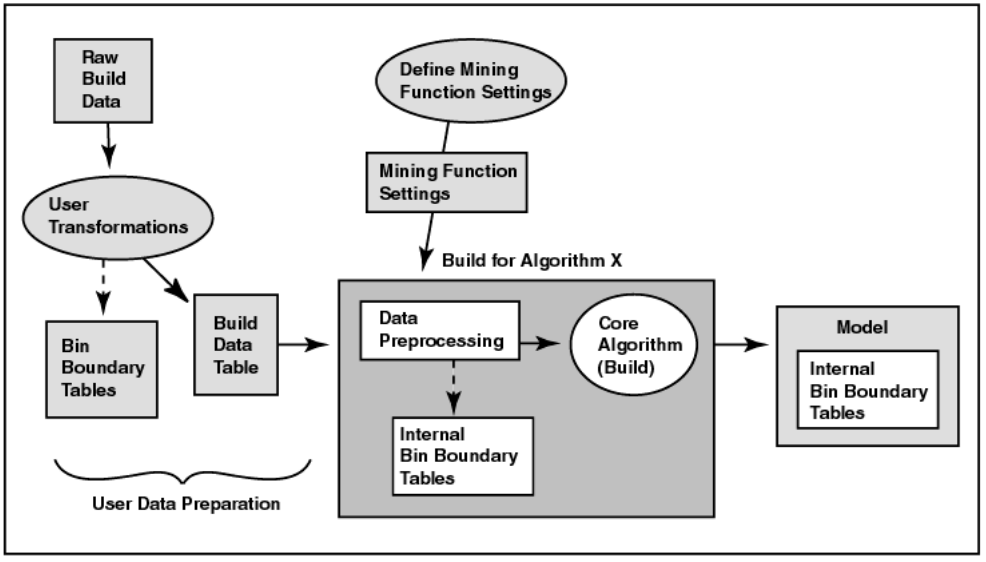
The typical steps for model building are as follows:

1. Specify input data by associating a physical data specification (see Section 1.6.1, "Physical Data Specification") with existing data, for example, a table or view, and prepare the data, including binning (see Section 1.7.2, "Discretization").
2. Create a mining function settings object, which specifies input parameters for 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).
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 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.

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.

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 test data must be in the same format and state of preprocessing as the data used to build 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 "churner," 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:

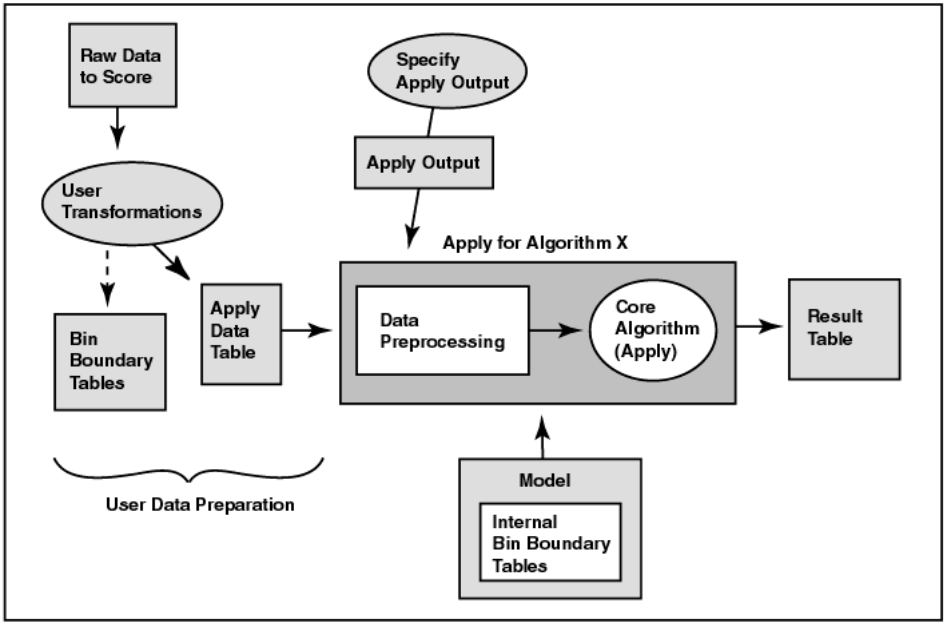
*targets_cumulative = lift_cumulative * percentage_records_cumulative*

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.

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, statistical analysis, 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 schema roles sequenceID, attribute_name, and value. We also refer to this format 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.

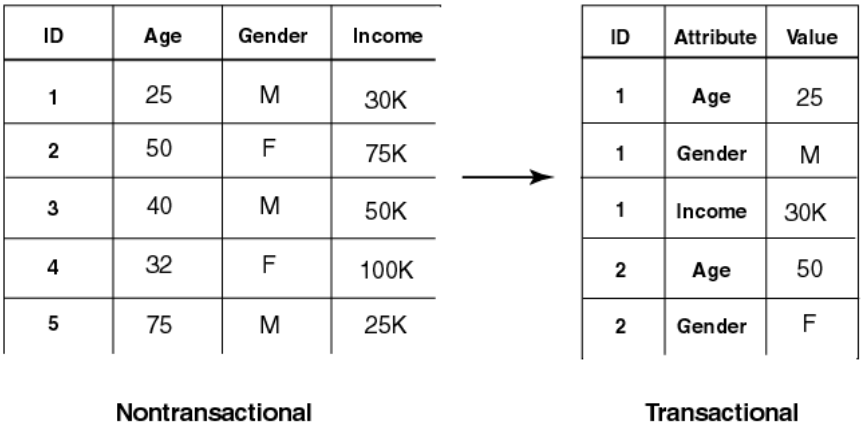
1.6.1.2 Nontransactional Data Format

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. We also refer to this format as *single-record case*.

Note that in Oracle 9*i*, ODM algorithms automatically convert all nontransactional data to transactional data prior to model building.

Figure 1–3 Nontransactional and Transactional Data Format



1.6.2 Mining Function Settings

A *mining function settings* (MFS) object captures the high-level parameters for building a mining model.

The mining function settings allow a user to specify the type of result desired without having to specify a particular algorithm. The ODM API allows a user to specify an algorithm; however, if none is specified, ODM system selects an algorithm based on the function settings specified.

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).

1.6.3 Mining Algorithm Settings

A *mining algorithm settings* object captures the parameters associated with a particular algorithm for building a model. It allows knowledgeable users to fine-tune the behavior of the algorithm. Generally, not all parameters must be specified. Missing parameters are replaced with system default values.

ODM's design, which separates mining algorithm settings from mining function settings, enables non-expert users to leverage ODM effectively, while expert users can have more control.

1.6.4 Logical Data Specification

A *logical data specification* (LDS) object is a set of mining attribute instances that describes the logical nature of the data used as input for model building. This set of mining attributes forms 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.

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.

The default usage for an attribute 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 extra attribute for use with results output during scoring)
- 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 treats column 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 underlying DMS. Some models can be used for direct inspection, for example, to examine the rules produced from association rules or clusters, others to score data 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.

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 space) for the result. The source data that is input to the apply function for scoring.

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

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

1.6.9 Confusion Matrix

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.

The row indexes of a confusion matrix correspond to *actual values* observed and 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. For example, a value of 25 for an actual value index of "churner" and a predicted value index of "nonchurner" indicates that the model incorrectly classified a "churner" as a "nonchurner" 25 times. A value of 516 for an actual / predicted value index of "churner" indicates that the model correctly classified a churner 516 times.

The predictions were correct 516 + 725 = 1241 times, and incorrect 25 + 10 = 35 times. The error rate is 35 / 1276 = 0.0274; the accuracy rate is 1241 / 1276 = 0.9725.

Figure 1–4 Confusion Matrix

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

The sum of the values in the matrix is equal to the number of scored records in the input data table.

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 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 data, the sequence ID associated with a given case

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

ODM supports renaming the resulting data columns with the source table column names.

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 for output to describe the output better (see source attribute, below). Batch scoring using an apply table results in a table called *apply output* table.

In individual record scoring, an input record is represented as an instance of *RecordInstance* object that contains a set of *AttributeInstance* objects, each of which describes the name of the attribute, the data type, and the value. Record scoring results in another instance of *RecordInstance.* 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 output table may contain columns copied directly from the input table. Similarly, the output record may contain attributes copied directly from the input record, although this is not particularly useful. These items are called *source attributes*, and each item is represented by an instance of *ApplySourceAttributeItem*. Source attributes can be used to identify the individual source records in the apply output and make the output more meaningful. There is no restriction on the number of source attributes to appear in 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. The number of predictions ranges from 1 to the total number of target values. One such item produces two columns (or attributes) in the output: prediction and optional probability, each of which is named by the user. There can be only one instance of *ApplyMultipleScoringItem* in a *MiningApplyOutput* object.
- **Multiple predictions based on target values:** An instance of *ApplyTargetProbabilityItem* results in predictions for a number N of target values. Each such target value must be one of the original target values used to build the model, and must be unique. One such item produces up to three columns (or attributes) in the output: *prediction*, *optional probability*, and *optional rank*, each of which is named by the user. There can be only one instance of *ApplyTargetProbabilityItem* in a *MiningApplyOutput* object, where

ApplyTargetProbabilityItem may not be used together with *ApplyMultipleScoringItem*.

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 probability 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. On the other hand, the number of rows in the output table is 10 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 and Discretization

In this section, we discuss ODM's handing of missing values and options for discretization (binning).

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.

1.7.2 Discretization

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

However, binning should be done with care, as it has a major impact on model accuracy. Manual binning by an expert, based on information about the data being binned and the problem being solved, can produce more accurate models.

ODM provides four ways to bin data:

- **Explicit specification:** Explicit specification of bin boundaries for a given attribute. The user provides one of the following:
  - for categorical data, sets of categories to be represented by a reduced cardinality value set
  - 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 automatically determines the `N` most frequent categories and puts all other categories 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 automatically 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.

  A model built using automated binning can score using that model, where the internal bin boundary tables are automatically applied to the score data. If the user preprocessed the data, whether or not automated binning was used, the user must preprocess the data before providing it to the algorithm.

Note: Current binning methods in ODM require 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.

1.8 PMML Support

The Predictive Model Markup Language (PMML) specifies data mining models using an XML DTD. 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 in order 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 ODM 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 a `MiningBuildTask` object.
5. After successful construction of the build task object, call a store method to store the 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, which stores the mining model.

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 model
3. Test model
4. Calculate lift

To implement a sequence of dependent task executions, you must 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 and evaluates and compares them 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 large data sets 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.

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.

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 3.1)
- `Sample_NaiveBayesApply_short.java` (Section 3.2)

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.3.1.

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.

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

3.1 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`. Note that this sample program does not use a property file.

3.1.1 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.

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.

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.

3.1.2 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?
- Should the build be done synchronously or asynchronously?

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.

3.1.3 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(();
```

3.1.4 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.

3.1.4.1 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.

3.1.4.2 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);
```

3.1.4.3 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);
```

3.1.5 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 data mining system is responsible for selecting the algorithm based on user-provided parameters.

3.1.5.1 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 DataPreparatioinStatus to unprepared.
//Automatic binning will be applied in this case.
ClassificationFunctionSettings m_ClassificationFunctionSettings =
         ClassificaitonFunctionSettings.create(
                  dmsConnection,
                  null,
                  m_PhysicalDataSpecification,
                  "class",
                  AttributeType.categorical,
                  DataPreparationStatus.getInstance("unprepared"));
```

3.1.5.2 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(
                 cmsConnection,
```

```
algorithmSetting,
m_PhysicalDataSpecification,
class,
Attribute Type.categorical,
DataPreparationStatus.getInstance(unprepared));
```

3.1.5.3 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_ClassiicationFunctionSettings.validate();
m_ClassificationFunctionSettings.store(dmsConnection, "Sample_NB_MFS");
```

3.1.6 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 build data is large, you will probably want to build the model asynchronously.

3.1.6.1 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");
```

3.1.6.2 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.

3.2 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.

3.2.1 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.

3.2.2 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?
- What model should be used for scoring, and should the scoring be done synchronously or asynchronously?

The following steps provide answers to the above questions:

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 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.

3.2.3 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((();
```

3.2.4 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.

3.2.4.1 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_APPLY_UNBINNED) and the schema_name for your installation
LocationAccessData lad =
    new LocationAccessData("CENSUS_APPLY_UNBINNED", "schema_name)";
```

Next, create the `PhysicalDataSpecification` instance.

3.2.4.2 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);
```

3.2.4.3 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);
```

3.2.5 Describe the Output Data

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

3.2.5.1 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");
```

3.2.6 Specify the Format of the Apply Output

The DMS also needs to know the format 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 `MiningApplyOutut` that you created.

3.2.6.1 Create an Empty Mining Apply Output Object

Create an empty `MiningApplyOutput` object as follows:

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

3.2.6.2 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 specifies 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);
```

3.2.6.3 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);
```

3.2.6.4 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();
```

3.2.7 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 is large, you will probably want to apply the model asynchronously.

3.2.7.1 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")
```

3.2.7.2 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 `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

The sample programs for ODM consist of Java classes and property files, along with the data required to run the programs. There are also scripts to compile and execute the sample programs. The sample programs and how to compile and execute them are briefly described in this appendix. The data used by the sample programs is installed when you install ODM.

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

A.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` on any system where ODM is installed.

A.2 List of ODM Sample Programs

ODM sample programs are provided to illustrate the features of ODM.

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; most sample programs have an (input) data set. 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. The rest of this section lists the ODM sample programs, arranged according to the ODM features that they illustrate.

A.2.1 Basic ODM Usage

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

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 Decision Tree Models

The following sample programs illustrate building a Decision Tree (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:

1. `Sample_NaiveBayesBuild.java`

- Property file: `Sample_NaiveBayesBuild.property`
- Data: `census_2d_build_unbinned`

2. `Sample_NaiveBayesLiftAndTest.java`
   - Property file: `Sample_NaiveLiftAndTest.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 and applying it:

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`

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 and attributes 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_test_unbinned and census_2d_apply_unbinned

A.3 Compiling and Executing ODM Sample Programs

This section provides a brief description of how to compile and execute the ODM sample programs. There are two cases:

- Compiling and executing the "short" programs Sample_NaiveBayesBuild_short.java and Sample_NaiveBayesApply_short.java
- Compiling and executing all other sample programs

A.3.1 Compiling and Executing the Short Sample Programs

Follow these steps to compile and execute the programs Sample_NaiveBayesBuild_short.java and Sample_NaiveBayesApply_short.java:

1. Install Oracle9*i* release 2 Enterprise Edition and the ODM 9.2.0 option. Ensure that you have a valid ORACLE_HOME environment variable setup.

   ODM depends on the following Oracle9*i* Java Archive files; ensure that they are in your CLASSPATH:

   ```
   $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
```

You may also need to include

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

in your CLASSPATH. See Section 2.1 for details.

2. The datasets used by the sample programs are installed during ODM installation. The default schema for these datasets is `odm_mtr`. If the default name is not correct for your installation, replace the schema name in the program.
3. Ensure you have installed JDK 1.3 or above and have a valid `JAVA_HOME` environment variable setup.
4. Before you execute either of these programs, make sure that you have specified the data mining server and the location access data appropriately for your installation.

   To specify the data mining server, substitute appropriate values for the italicized items in the following line:

   ```
   dms = new DataMiningServer(DB_URL, user_name, password);
   ```

   To specify location access data, substitute appropriate values for the italicized items in the following line:

   ```
   LocationAccessData("CENSUS_2D_BUILD_UNBINNED", schema_name);
   ```

   For `Sample_NaiveBayesApply_short.java`, you must also specify a location for the output table; substitute appropriate values for the italicized item in the following line:

   ```
   LocationAccessData ladOutput =
       new LocationAccessData("CENSUS_NB_APPLY_RESULT", output_schema_name)
   ```

5. The ODM sample programs include scripts that compile the sample programs. Compile an ODM sample program by running the appropriate script.

   - On UNIX platforms, use

     ```
     /usr/bin/sh compileSampleCode.sh program-name
     ```

   - On Windows platforms, use

```
compileSampleCode.bat program-name
```

6. Before you run a sample program, verify that the ODM monitor is running. If you need to start the monitor, log in to the ODM schema and type

   ```
   exec odm_start_monitor
   ```

7. Execute the sample program as you would execute any Java program.

   Note: Since these short programs do not use `Sample_Global.property`, you cannot execute them using the `executeSampleCode` scripts.

8. You must perform cleanup before you execute either of these programs a second time.

A.3.2 Compiling and Executing All Other ODM Sample Programs

Follow these steps to compile all of the sample programs that use `Sample_Global.property`:

1. Install Oracle9*i* release 2 Enterprise Edition and the ODM 9.2.0 option. Ensure that you have a valid `ORACLE_HOME` environment variable setup.

   ODM depends on the following Oracle9*i* Java Archive files; ensure that they are in your `CLASSPATH`:

   ```
   $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
   ```

   You may also need to include

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

   in your CLASSPATH. See Section 2.1 for details.

2. Modify the `Sample_Global.property` file to replace generic placeholders with the details for your database installation. Mining Server details must point to the schema where the Mining Server is installed on your system.

You must replace the following tags: MyHost, MyPort, MySid (the SERVICE_NAME for your database), MyName (the default is ODM), and MyPW (the default is ODM).

For example:

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

3. The datasets used by the sample programs are installed during ODM installation. The default schema for these datasets is odm_mtr. If the default name is not correct for your installation, replace the schema name in Sample_Global.property.
4. Ensure you have installed JDK 1.3 or above and have a valid JAVA_HOME environment variable setup.
5. Edit the code settings in the property file for the program that you wish to compile and execute. For example, if you plan to compile and execute Sample_ModelSeeker.java, you should edit Sample_ModelSeeker.property.
6. The ODM sample programs include scripts that compile the sample programs. Compile an ODM sample program by running the appropriate script.
   - On UNIX platforms, use

     ```
     /usr/bin/sh compileSampleCode.sh program-name
     ```

     For example, to compile Sample_ModelSeeker.java, type

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

   - On Windows platforms, use

     ```
     compileSampleCode.bat program-name
     ```

     For example, to compile Sample_ModelSeeker.java, type

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

7. Before you run a sample program, verify that the ODM monitor is running. If you need to start the monitor, log in to the ODM schema and type

   ```
   exec odm_start_monitor
   ```

8. The ODM sample programs include scripts that execute the sample programs. Execute an ODM sample program by running the appropriate script.

   - On UNIX platforms, use

     ```
     /usr/bin/sh executeSampleCode.sh classname [property_ file]
     ```

     For example, to execute `Sample_ModelSeeker.java` with the property file `myFile.property`, type

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

   - On Windows platforms, use

     ```
     executeSampleCode.bat classname [property_ file ]
     ```

     For example, to execute `Sample_ModelSeeker.java` with the property file `myFile.property`, type

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

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 decision trees / 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

Association rules capture 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 the 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 is 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 of 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.

centroid

See **cluster centroid**.

classification

The process of predicting the unknown value of the target attribute for new records using a model built from records with known target values. ODM supports two algorithms for classification, Naive Bayes and decision trees / Adaptive Bayes Networks. You can use ODM Model Seeker to find a "best" classification model.

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 technique 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. The row indexes of a confusion matrix correspond to *actual values* observed and 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.

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.

cross validation

A method of evaluating the accuracy of a classification or regression model. 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, 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

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

discretization

Discretization groups related values together, which 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).

DMS

See **data mining server**.

feature

A 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*}}.

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 underlying 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

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 methods vary in the way they treat missing values. Typically, they ignore the missing values, or omit any records containing missing values, or replace missing values with the mode or mean, or infer missing values from existing values.

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**.

nontransactional format

Each case in the data is stored as one record (row) in a table. 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 are continuous, as opposed to discrete or categorical 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, data items that fall outside the boundaries that enclose most other data items 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 *X* may be a compound predicate.

settings

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

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.

target

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

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 schema roles `sequenceID`, `attribute_name`, and `value`. Also known as *multi-record case*.

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. In supervised learning, this correct result is provided in the target attribute. Unsupervised learning has no such target attribute. Clustering and association rules are unsupervised.

Index

M

N

O

P

R

S

T

U