Close Window

Print Story

Using Java Data Mining to Develop Advanced Analytics Applications

With the standardization of the Java Data Mining (JDM) API, Enterprise Java applications have been given predictive technologies.

Data mining is a widely accepted technology used for extracting hidden patterns from data. It is used to solve many business problems like identifying cross-sell or up-sell opportunities for specific customers based on customer profiles and purchase patterns, predicting which customers are likely to churn, creating effective product campaigns, detecting fraud, and finding natural segments.

More and more data mining algorithms are being embedded in databases. Advanced analytics, like data mining, is now widely integrated with applications. The objective of this article is to introduce Java developers to data mining and explain how the JDM standard can be used to integrate this technology with enterprise applications.

Data Mining Functions
Data mining offers different techniques, *aka mining functions*, that can be used depending on the type of problem to be solved. For example, a marketing manager who wants to find out which customers are likely to buy a new product can use the classification function. Similarly a supermarket manager who wants to determine which products to put next to milk and eggs, or what coupons to issue to a given customer to promote the purchase of related items can use the association function.

Data mining functions are divided into two main types called *supervised* (directed) and *unsupervised* (undirected).

Supervised functions are used to predict a value. They require a user to specify a set of predictor attributes and a target attribute. Predictors are the attributes used to predict the target attribute value. For example, a customer's age, address, occupation, and products purchased can be used to predict the target attribute "Will the customer buy the new product? (YES/NO)."

Classification and regression are categorized as supervised functions. Classification is used to predict discrete values, e.g., "buy" or "notBuy," and regression is used to predict continuous values, e.g., salary or price.

Unsupervised functions are used to find the intrinsic structure, relations, or affinities in data. Unsupervised mining doesn't use a target. Clustering and association functions come under this category. Clustering is used to find the natural groupings of data, and association is used to infer co-occurance rules from the data.

The Data Mining Process
Typically data mining projects are initiated by a business problem. For example, a CEO could ask, "How can

I target the right customers to maximize profits?" Once the business problem is defined, the next step is to understand the data available and select the appropriate data to solve the problem. Based on the data's characteristics, prepare the data for mining. Select the right mining function and build a mining model with the data. After building the model, evaluate the model results. After evaluation, deploy the model. The CRISP-DM standard details the typical data mining process. Figure 1 illustrates a typical data mining process.

Enterprise applications like CRM analytics try to automate the data-mining process for common problems like intelligent marketing campaigns and market-basket analysis.

JDM API Overview

The Java Community Process (JCP) released the JDM 1.0 standard in August of 2004. JDM provides an industry standard API to integrate data mining functionality with applications. It facilitates the development of vendor-neutral data mining tools/solutions. It supports many commonly used mining functions and algorithms.

JDM uses the factory-method pattern to define Java interfaces that can be implemented in a vendor-neutral fashion. In the analytics business there's a broad set of data mining vendors who sell everything from a complete data mining solution to a single mining function. JDM conformance states that even a vendor with one algorithm/function can be JDM-conformant.

In JDM, *javax.datamining* is the base package that defines infrastructure interfaces and exception classes. Sub-packages are divided by mining function type, algorithm type, and core sub-packages. Core subpackages are *javax.datamining.resource*, *javax.datamining.base*, *javax.datamining.data*. The resource package defines connection-related interfaces that enable the applications to access Data Mining Engine (DME). The base package defines prime objects like mining model. The data package defines all physical and logical data-related interfaces. The *javax.datamining.supervised* package defines the supervised function-related interfaces and the *javax.datamining.algorithm* package contains all mining algorithm subclass packages.

Solving the Customer Churn Problem Using JDM

Problem Definition

Customer attrition is one of the big problems companies face. Knowing which customers are likely to leave can be used to develop a customer-retention strategy. Using data-mining classifications one can predict which customers are likely to leave. In the telecommunications industry, this problem is known as customer churn. Churn is a measure of the number of customers who leave or switch to competitors.

Understand and Prepare the Data

Based on the problem and its scope, domain experts, data analysts, and database administrators (DBA) will be involved in understanding and preparing data for mining. Domain experts and data analysts specify the data required for solving the problem. A DBA collects it and provides it in the format the analyst asked for.

In the following example, several customer attributes are identified to solve the churn problem. For simplicity's sake, we'll look at 10 predictors. However, a real-world dataset could have hundreds or even thousands of attributes (see Table 1).

Here CUSTOMER_ID is used as the case id, which is the unique identifier of a customer. The CHURN column is the target, which is the attribute to be predicted. All other attributes are used as predictors. For each predictor, the attribute type needs to be defined based on data characteristics.

There are three types of attributes, i.e., categorical, numerical, and ordinal.

A categorical attribute is 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 and gender, or ordered (ordinal) such as high, medium, or low temperatures. A numerical attribute is an attribute whose values are numbers that are either integer or real. Numerical attribute values are continuous as opposed to discrete or categorical values.

For supervised problems like this, historical data must be split into two datasets, i.e., for building and testing the model. Model building and testing operations need historical data about both types of customers, i.e., those who already left and those who are loyal. The model-apply operation requires data about new customers, whose churn details are to be predicted.

The Data Mining Engine (DME)
In JDM, the Data Mining Engine (DME) is the server that provides infrastructure and offers a set of data mining services. Every vendor must have a DME. For example, a database vendor providing embedded data-mining functionality inside the database can refer to the database server as its data mining engine.

JDM provides a Connection object for connecting to the DME. Applications can use the JNDI service to register the DME connection factory to access a DME in a vendor-neutral approach. The *javax.datamining.resource.Connection* object is used to represent a DME connection, do data-mining operations in the DME, and get metadata from the DME.

Listing 1 illustrates how to connect to a DME using a connection factory that's registered in a JNDI server.

Describe the Data
In data mining, the ability to describe the physical and logical characteristics of the mining data for building a model is important.

JDM defines a detailed API for describing physical and logical characteristics of the mining data.

The *javax.datamining.data.PhysicalDataSet* object is used to encapsulate location details and physical characteristics of the data.

The *javax.datamining.data.LogicalData* object is used to encapsulate logical characteristics of the data.

A logical attribute can be defined for each physical attribute in the physical data set. A logical attribute defines the attribute type and the data preparation status. The data preparation status defines whether the data in a column is prepared or not prepared. Some vendors support internal preparation of the data. In JDM, the physical data set and logical data are named objects, which can be stored in and retrieved from the DME.

Listing 2 illustrates how to create a *PhysicalDataSet* object and save it in the DME. Here *PhysicalDataSet* encapsulates "CHURN_BUILD_TABLE" details. In this table "CUSTOMER_ID" is used as the caseId.

Listing 3 illustrates how to create a *LogicalData* object and save it in the DME. Here *LogicalData* is used to specify the attribute types. Some vendors derive some of the logical characteristics of attributes from the physical data. So JDM specifies logical data as an optional feature that vendors can support. Logical data is an input for the model-build operation. Other operations like apply and test get this information from the model.

Build the Mining Model

One important function of data mining is the production of a model. A model can be supervised or unsupervised.

In JDM *javax.datamining.base.Model* is the base class for all model types. To produce a mining model, one of the key inputs is the build settings object.

javax.datamining.base.BuildSettings is the base class for all build-settings objects, it encapsulates the algorithm settings, function settings, and logical data. The JDM API defines the specialized build-settings classes for each mining function.

In this example, the *ClassificationSettings* object is used to build a classification model to classify churners.

Applications can select an algorithm that works best for solving a business problem. Selecting the best algorithm and its settings values requires some knowledge of how each algorithm works and experimentation with different algorithms and settings. The JDM API defines the interfaces to represent the various mining algorithms.

In this example, we will use the decision-tree algorithm.

In JDM, *javax.datamining.algorithm.tree.TreeSettings* object is used for representing decision-tree algorithm setting. Some vendors support implicit algorithm selection based on function and data characteristics. In those cases, applications can build models without specifying the algorithm settings.

Listing 4 illustrates how to create classification settings and save them in the DME. Here a classification-settings object encapsulates the logical data, algorithm settings, and target attri-bute details to build the churn model. Here the decision-tree algorithm is used. For more details about the algorithm settings refer to the JDM API documentation.

Listing 5 illustrates how to build a mining model by executing the build task. Typically model building is a long-running operation, JDM defines a task object that encapsulates the input and output details of a mining operation. A task object can be executed asynchronously or synchronously by an application. Applications can monitor the task-execution status using an execution handle.

An execution-handle object is created when the task is submitted for execution. For more details about the task execution and the execution handle, refer to the JDM API documentation.

Here the build task is created by specifying the input physical dataset name, build settings name, and output model name. The build task is saved and executed asynchronously in the DME. Applications can either wait for the task to be completed, or execute the task and check the status later.

Test the Mining Model

After building a mining model, one can evaluate the model using different test methodologies. The JDM API defines industry standard testing methodologies for supervised models.

For a classification model like the churn model, the *ClassificationTestTask* is used to compute classification test metrics. This task encapsulates input model name, test data name, and metrics object name. It produces a *ClassificationTestMetrics* object that encapsulates the accuracy, confusion matrix, and lift metrics details that are computed using the model.

Accuracy provides an estimate of how accurately the model can predict the target. For example, 0.9 accuracy means the model can accurately predict the results 90% of the time.

Confusion matrix is a two-dimensional, N x N table that indicates the number of correct and incorrect predictions a classification model made on specific test data. It provides a measure of how well a classification model predicts the outcome and where it makes mistakes.

Lift is a measure of how much better prediction results are using a model as opposed to chance. To explain the lift we will use a product campaign example. Say product campaigning to all 100,000 existing customers results in sales of 10,000 products. However, by using the mining model say we sell 9,000 products by campaigning to only 30,000 selected customers. So by using the mining model, campaign efficiency is increased three times, so the lift value is computed as 3, i.e., (9000/30000)/(10000/100000).

Listing 6 illustrates how to test the churn model by executing the classification test task using "CHURN_TEST_TABLE." After successfully completing the task, a classification test metrics object is created in the DME. It can be retrieved from the DME to explore the test metrics. (Listings 6-8 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

Apply the Mining Model
After evaluating the model, the model is ready to be deployed to make predictions. JDM provides an *ApplySettings* interface that encapsulates the settings related to the apply operation. The apply operation will result in an output table with the predictions for each case. Apply settings can be configured to produce different contents in the output table. For more details on apply settings, refer to JDM API documentation.

In this example, we use the top prediction apply setting to produce the top prediction for each case. The *DataSetApplyTask* is used to apply the churn model on the "CHURN_APPLY_TABLE." JDM supports *RecordApplyTask* to compute the prediction for a single record; this task is useful for real-time predictions. In this example, we use the dataset apply task to do the batch apply to make predictions for all the records in the "CHURN_APPLY_TABLE".

Listing 7 illustrates how to apply the "CHURN_MODEL" on "CHURN_APPLY_TABLE" to produce an output table "CHURN_APPLY_RESULTS" that will have the predicted churn value "YES or NO" for each customer.

After doing the apply task, a "CHURN_APPLY_RESULTS" table will be created with two columns, "CUSOMER_ID" and "PREDICTED_CHURN." The probability associated with each prediction can be obtained by specifying it in the *ApplySettings*.

Here the *mapTopPrediction* method is used to map the top prediction value to the column name. The source destination map is used to carry over some of the columns from the input table to the apply-output table along with the prediction columns. In this case, "CUSTOMER_ID" column is carried over from the apply-input table to the output table. JDM specifies many other output formats so applications can generate the apply-output table in the required format. A discussion of all the available options is beyond the scope of this article.

Figure 2 summarizes the JDM data mining process flow that we did in this example.

Market Basket Analysis Example

To explain the use of unsupervised data mining in a practical scenario, we'll use one of the most popular data mining problems called *market basket analysis*.

The purpose of market basket analysis is to determine what products customers buy together. Knowing what products people buy together can be helpful to traditional retailers and web stores like Amazon.

The information can be used to design store layouts, web page designs, and catalog designs by keeping all cross-sell and up-sell products together. It can also be used in product promotions like discounts for cross-sell or up-sell products. Direct marketers can use basket analysis results to decide what new products to offer their prior customers.

To do market basket analysis, it's necessary to list the transactions customers made. Sometimes customer demographics and promotion/discount details are used to infer rules related to demographics and promotions. Here we use five transactions at a pizza store. For simplicity's sake, we'll ignore the demographics and promotion/discount details.

Transaction 1: Pepperoni Pizza, Diet Coke, Buffalo wings
Transaction 2: Buffalo wings, Diet Coke
Transaction 3: Pepperoni Pizza, Diet Coke
Transaction 4: Diet Coke, French Fries
Transaction 5: Diet Coke, Buffalo wings

The first step is to transform the transaction data above into a transactional format, i.e., a table with transaction id and product name columns. The table will look like Table 2. Only the items purchased are listed.

An association function is used for market basket analysis. An association model extracts the rules stating the support and confidence in each rule. The user can specify the minimum support, minimum confidence, and maximum rule length as build settings before building the model.

Since we have only five transactions, we'll build a model to extract all the possible rules by specifying minimum support as 0.1, minimum confidence as 0.51, and no maximum limit for the rule length. This model produces five rules (see Table 3).

In a typical scenario, you may have millions of transactions with thousands of products, so understanding the support and confidence measures and how these are calculated provides good insight into which rules need to be selected for a business problem.

Support is the percentage of records containing the item combination compared to the total number of records. For example take Rule 1, which says, "If Buffalo wings are purchased then diet coke will also be purchased." To calculate the support for this rule, we need to know how many of the five transactions conform to the rule. Actually, three transactions, i.e., 1, 2 and 5, conform to it. So the support for this rule is 3/5=0.6.

Confidence of an association rule is the support for the combination divided by the support for the condition. Support gives an incomplete measure of the quality of an association rule. If you compare Rule 1 with Rule 5, both of them have the same support, i.e., 0.6, because support is not directional. Confidence is directional, so that makes Rule 1 a better rule than Rule 5.

Rule length can be used to limit the length of the rules. When there are thousands of items/products with millions of transactions, rules get complex and lengthy, so it's used to limit the length of the rules in a model.

Using JDM to Solve the Market Basket Problem

So how does one use JDM API to build an association rules model and extract the appropriate rules from the model?

Typically data for association rules will be in a transactional format. A transactional format table will have three columns: "case id", "attribute name," and "attribute value" columns.

In JDM by using the *PhysicalAttributeRole* enumeration, the transactional format data can be described. The *AssociationSettings* interface is used to specify the build settings for association rules. It has minimum support, minimum confidence, and maximum rule-length settings that can be used to control the size of association rules model.

Listing 8 illustrates building a market-basket analysis model using the JDM association function and exploring the rules from the model using rule filters.

Conclusion

The use of data mining to solve business problems is on the upswing. JDM provides a standard Java interface for developing vendor-neutral data-mining applications. JDM supports common data-mining operations, as well as the creation, persistence, access, and maintenance of the metadata supporting mining activities. Oracle initiated a new JSR-247 to work on new features for a future version of the JDM standard.

References

- Java Data Mining Specification. http://jcp.org/aboutJava/communityprocess/final/jsr073/index.html
- Java Data Mining API Javadoc.www.oracle.com/technology/products/bi/odm/JSR-73/index.html
- Java Data Mining Project Home. https://datamining.dev.java.net
- Cross-Industry Standard Process for Data Mining (CRISP-DM). www.crisp-dm.org
- JSR-247. http://jcp.org/en/jsr/detail?id=247