EXPERIMENTAL STUDY FOR EXTENDING DATA MINING STANDARDS

AHMET DAYLAN

BOĞAZİÇİ UNIVERSITY

2008

EXPERIMENTAL STUDY FOR EXTENDING DATA MINING STANDARDS

Thesis submitted to the

Institute for Graduate Studies in the Social Sciences

in partial fulfillment of the requirements for the degree of

Master of Arts

in

Management Information Systems

by

AHMET DAYLAN

Boğaziçi University

2008

Experimental Study for Extending Data Mining Standards

The thesis of Ahmet Daylan

has been approved by

Asst. Prof. Bertan Badur _____
(Thesis advisor)

Assoc. Prof. Osman N. Darcan _____
(Thesis advisor)

Asst. Prof. Sona Mardikyan _____

Assoc. Prof. Eyüp Çetin _____

Asst. Prof. U.Tuğba Şimşek _____

September 2008

Thesis Abstract

Ahmet Daylan, “Experimental Study for Extending Data Mining Standards”

Data mining is becoming a mainstream technology used in business intelligence solutions supporting various industries and lines of business. Although there are plenty of data mining products at the market, these products are difficult to integrate with user applications due to the lack of standardization protocols. Conforming to common standards facilitates development, implementation and maintenance of applications as well as communication among them. In addition, these standards enable data miners to develop data mining process easily. As being one of the well-established data mining standards, Java Data Mining (JDM) allows Java applications to communicate with data mining engines to build, test and apply mining models. First release of JDM supports the basic data mining functionalities; classification, regression, attribute importance, clustering, and association. Currently developing version (JDM 2.0) proposes the additional set of functionalities such as time series, feature extraction, text mining. In this study, we present the experimental framework developed to extend the JDM 2.0 by including ensembles methods such as boosting, bagging to improve the classification accuracy and bootstrap to assess accuracy. We have applied our extended JDM functionalities with two well known datasets that are iris dataset and Sales History (SH) dataset.

Tez Özeti

Ahmet Daylan, “Veri Madenciliği Standartlarının Genişletilmesi”

Veri madenciliği, iş zekası çözümlerinde çeşitli endüstri ve iş kollarını destekleyen ana teknoloji olarak kullanılmaya başlanıyor. Birçok veri madenciliği ürününün piyasada olmasına rağmen, standart protokollerin olmamasından dolayı bu ürünler uygulamalar ile entegre edilememektedir. Genel standartlara uymak, uygulama geliştirmeyi, bakımını ve uygulamalar arasındaki iletişimi kolaylaştırır. Ayrıca, bu standartlar veri madenciliği sürecinin daha kolay yapılabilmesine olanak verir. İleri seviye veri madenciliği standartlarından bir tanesi olan Java Data Mining (JDM) ile Java uygulamaları veri madenciliği motorları ile iletişim kurarak modellerin yaratılması, test edilmesi ve uygulanması işlemleri yapılabilir. JDM’in ilk versiyonu ile sınıflandırma, regresyon, öbekleme ve eşleştirme gibi temel veri madenciliği fonksiyonlarını yapılabilmektedir. Gelişmekte olan JDM 2.0 versiyonu ek gelişmiş bazı veri madenciliği fonksiyonları önermektedir; zaman serileri, özellik çıkarma ve metin madenciliği. Bu çalışmada JDM 2.0 standartlarını genişletmek üzere geliştirdiğimiz deneysel çerçeveyi sunmaktayız. Bu çerçevede, Bootstrap ile tahmin doğruluğunu belirleme, Bagging ve Boosting ile tahmin doğruluğunu artırma çalışmaları bulunmaktadır. Genişlettiğimiz JDM fonksiyonlarını iyi bilinen iris ve Sales History (SH) verisetleri üzerinde uyguladık.

ACKNOWLEDGEMENTS

The first, I am grateful to my thesis supervisors Asst. Prof. Bertan Badur and Assoc. Prof. Osman N. Darcan for their guidance and patience. I also wish to thank Asst. Prof. Sona Mardikyan, Assoc. Prof. Eyüp Çetin and Asst. Prof. Tuğba Şimşek my thesis committee members, who devoted their valuable time and energy to this study.

I will also never forget the encouragements, helps and supports of all of my friends who are always with me whenever I need them.

CONTENTS

TABLES

FIGURES

CHAPTER I

INTRODUCTION

Data mining is viewed as a natural evolution of information technology where knowledge is extracted from huge amount of data accumulated in databases and data warehouses. When successfully applied, data mining provide comparative advantages in industries and lines of business. Therefore, plenty of data mining products have evolved at the market as a part of business intelligence solutions. Without widely-accepted standards, advantages of data mining solutions cannot be fully utilized. Hence, many data mining standards are developed in order to facilitate development, implementation and maintenance of applications as well as communication among them. Conforming to data mining standards can improve data mining process in terms of reduction of costs and completion time and flexibility so as to adapt to a rapidly changing environment.

Java Data Mining (JDM) is one of the well-established data mining standards (JSR-73, JSR-247). It allows Java applications to communicate with data mining engines to build, test and apply mining models. First release of JDM (JDM 1.0) supports the basic data mining functionalities as classification, regression, attribute importance, clustering, association and anomaly detection. JSR-247 Expert group is working on the second release of JDM that extends the set of functionalities supported by JDM 1.0. Although, measuring the accuracy of classification models is crucial for the success of a data mining study, it is not supported in the currently developing version of JDM. In addition, improving the accuracy of classification

models by combining different classifiers is also essential to increase the utilization of these models. This facility is not included in JDM 2.0 either.

In this study, we developed an experimental framework to extend the scope of JDM 2.0 by including sampling methods to evaluate the accuracy of data mining models such as bootstrapping as well as including ensemble methods to improve the predicting power of regression and classification models such as boosting and bagging. We have applied our extended JDM functionalities with two well known datasets that are iris dataset (Asunction and Newman, 2007) and sales history dataset (Sales History). Bootstrap algorithm applied and accuracy values of the models on datasets are measured. Performance of bagging and boosting algorithms on predicting the target attributes are compared.

The outline of this study is as follows. In chapter 2, data mining and its basic functionalities are presented together with emphasize to bootstrap, bagging and boosting. In chapter 3 various standards related to data mining are discussed and JDM technology is introduced in detail. Chapter 4 covers our suggested design as improvements over JDM technology standards. Followed by the applications of JDM improvements to the iris dataset and sales history dataset are given in chapter 5. Finally, chapter 6 summarizes our work and presents directions for further research.

CHAPTER II

DATA MINING

Data mining is the process of analyzing data from different perspectives and summarizing it into useful information mainly used in order to increase revenue or to cut costs. In relation to enterprise resource planning, data mining is the statistical and logical analysis of large sets of transaction data, looking for patterns that can aid decision making (Ellen Monk and Bret Wagner, 2006). The phrase *data mining* is used interchangeably with *Knowledge Discovery in Databases (KDD),* although data mining is treated as simply a significant step in the process of KDD. It consists of an iterative sequence of the following steps; data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation and knowledge presentation (Han and Kamber, 2005).

The traditional method of turning data into knowledge relies on manual analysis and interpretation. For example, in the health-care industry, it is common for specialists to periodically analyze current trends manually in health-care data. The specialists then provide a report detailing the analysis to the sponsoring health-care organization; this report becomes the basis for future decision making and planning for health-care management. For these applications, this form of manual probing of a data set is slow, expensive, and highly subjective. As data volumes grow dramatically, huge datasets are stored in commercial or non commercial automated relational databases. Relational databases comprise one or more tables which are two dimensional arrays containing single-value entries and no duplicate rows (Kroenke,

2006). Values for a given entity are shown in rows; values of attributes of that entity are shown in columns. There are attributes used commonly referred to as a column and there are records basically the same thing as a row or tuple in databases. Today, databases are increasing in size in two ways: (1) the number N of records or objects in the database and (2) the number d of fields or attributes to an object. Under these conditions, manual data analysis is becoming completely impractical in many domains.

Data mining has evolved, and continues to evolve, from the intersection of research fields such as machine learning, pattern recognition, database systems, statistics, artificial intelligence, knowledge acquisition for expert systems, data visualization, and high-performance computing. The unifying goal is extracting high-level knowledge from low-level data in the context of large data sets. The data mining process is interactive and iterative, involving numerous steps with many decisions made by the user. (Brachman and Anand 1996) give a practical view of the data mining process, emphasizing the interactive nature of the process.

Data mining process comprises many challenges. There are several aspects of data that make it a challenge to mine. Data is usually available in the raw form such as noisy images, as structured or unstructured formats. As a result, a substantial amount of pre-processing time is needed to bring this raw data into a form suitable for the detection of patterns or useful information. Data can also be high-dimensional, where multiple features or characteristics are used to represent the objects or structures of interest. It often has both a spatial and a temporal component. Incomplete, noisy and inconsistent data are commonplace properties of large real-

world databases and data warehouses. Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because it was not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding or because of equipment malfunctions. Data inconsistent with other data may have been deleted. Furthermore, the recording of the history or modifications to the data may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred. There are many reasons for noisy data (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. There may be technology limitations, such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes used. Duplicate tuples also require data cleaning. Detecting data anomalies, rectifying them early and reducing the data be analyzed can lead to huge payoffs for decision making (Han and Kamber, 2005). Data mining tools by applying data mining functionalities can answer business and scientific questions that traditionally were too time-consuming to resolve. Scalable and efficient mining algorithms scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations. Other aspects of discovered data are usefulness, certainty and expressiveness of data mining results, expression of various kinds of data mining results, interactive mining knowledge at multiple abstraction levels, mining information from different sources of data, protection of privacy and data security (Han, Chen and Yu, 2000). These challenges may be carrying conflicting goals. For

example, the goal of protection of data security may conflict with the requirement of interactive mining of multiple - level knowledge from different angles.

Data mining functionalities are classified into two broad categories as descriptive and predictive ones (Han and Kamber, 2005). Descriptive functionalities help understanding characteristics of data in databases. Description focuses on finding human-interpretable patterns describing the data. Characterization, association analysis, clustering are examples of descriptive functionalities. *Characterization* is a summarization of the general features of a target class of data. *Association analysis* is the discovery of rules in transactional databases. It is widely used for market basket analysis to uncover the items that are purchased together. For example, the presence of one set of items implies the presence of another item or set of items, such as 90 percent of the people who buy milk and eggs also buy bread in the same transaction. *Clustering analysis* identifies clusters embedded in the data, where a cluster is a collection of data objects that are similar to one another. Clustering is used in customer segmentation, gene and protein analysis, product grouping, finding taxonomies and text mining. Typical goals for clustering can include finding representative cases from a large dataset to support data reduction, identifying natural clusters in a dataset and finding data either does not belong to any of the found clusters or belongs to a cluster of a few cases providing a kind of outlier or anomaly detection.

On the other hand, predictive functionalities make predictions based on inferences on databases. Predictive functionalities generally are based on models which are simplified abstract views of the complex reality. There are quantitative

models such as classification, regression and time series which are examples of predictive functionalities. Prediction involves using some variables or fields in the database to predict unknown or future values of other variables of interest. *Classification* is the process of finding a set of models that describe and distinguish data using a training dataset. The derived model is used to predict class labels that are unknown. Classification can be used to make predictions involving a wide range of problems, including campaign response, customer churn modeling, customer attrition modeling, and credit analysis. *Regression analysis* is similar to classification in terms of predicting a target variable. In this functionality the target variable is continuous rather than categorical for example in financial forecasting, time series prediction, house pricing, customer lifetime value prediction and environmental modeling such as predicting levels of oxygen in the atmosphere. Although the boundaries between prediction and description are not sharp (some of the predictive models can be descriptive, to the degree that they are understandable, and vice versa), the distinction is useful for understanding the overall discovery goal. *Time Series* is performed on time-series databases and sequence databases that consist of sequence of values or events changing with time. Trend analysis, similarity search and mining of sequential patterns and periodic patterns are some aspects of time series.

For predictive functionalities, accuracy of the algorithm is especially important so as to increase utilization of the model. The basic samples of data needed to calculate the confidence intervals have distributions which depart from the traditional parametric distributions. Thus, classical hypothesis-testing procedures based on strong parametric assumptions cannot be used to estimate the confidence intervals. In order to obtain results as reliable as possible, a statistical technique

which is applicable regardless of the form of the data probability density function has to be utilized. In other words, this method should make no assumption about the different data distributions (Efron and Tibshhirani, 1993). In order to evaluate accuracy values of models, bootstrap method is used. Bagging and boosting are the examples of techniques (ensemble methods) for increasing predictive accuracy of models.

Bootstrap

Bootstrap is a common technique for assessing accuracy of predictive models based on randomly sampled partitions of data (Efron and Tibshhirani, 1993). Bootstrap method samples the given training dataset uniformly with replacement. A dataset of *n* instances is sampled *n* times, with replacement, to give another dataset of *n* instances. Since some elements in this second dataset will almost certainly be repeated, there must be some instances in the original dataset that have not been picked: these are used as test instances. Sampling procedure is repeated *k* times, where in each iteration; accuracy of the model is obtained by the combination of the accuracy values of training dataset and test dataset. Since the chance of an instance being picked for the training set is 0.632 (Alpaydın, 2004), the overall accuracy of the model *Acc(M)* is then estimated as in (1). Where $\text{Acc(M)}_{\text{test\_set}}$ is the accuracy of the test dataset and $\text{Acc(M)}_{\text{training\_set}}$ is the accuracy of the training dataset.

$$Acc(M) = \frac{\sum_{1}^{k} (0.632 Acc(M_i)_{test\_set} + 0.368 Acc(M_i)_{training\_set})}{k} \quad (1)$$

Bagging

Bagging algorithm is the concept of combining the predicted classifications from multiple models, or from the same type of model for different learning data. In other words, bagging combines a series of *k* learned models with the aim of creating an improved composite model. For example, suppose a patient would like to have a diagnosis based on the symptoms. Instead of asking one doctor, patient prefers to ask several doctors. If a certain diagnosis occurs more than any others, this is chosen as the final and the best diagnosis. The final diagnosis is based on majority of votes, where each doctor gets equal weight. This is the basic idea behind bagging which is an acronym for Bootstrap AGGregatING (Breiman 1996). Algorithm for bagging is shown in Figure 1. *n* is the number of instances in the training data. *k* is the number of iterations. For each iteration, *n* instances are sampled from training data with replacement and learning algorithm is applied to the sample. The iteration generates *k* models. During the classification of instances, class of a certain instance is predicted using all of the models. In other words, every instance has a number of *k* estimates which were performed by *k* stored models. Bagging algorithm returns the class that is predicted most often as models get equal vote. Some application areas of bagging technique are forecasting economic time series (Inoue, 2007) and predicting financial crisis in emerging markets (Apoteker, 2005).

Model generation
Let n be the number of instances in the training data
For each of k iterations:
    Sample n instances with replacement from training data.
    Apply the learning algorithm to the sample.
    Store the resulting model

Classification
For each of the k models:
    Predict class of instance using model.
Return class that has been predicted most often

Figure 1. Algorithm for bagging.

Boosting

Boosting algorithm is a learning algorithm that iteratively uses samples and strengthens its estimate accuracy by combining previous models (Freund, 1997). Many variants of this algorithm can be found where methodology of assigning new weights and sampling approach differs. One of the most popular implementations of boosting algorithm is the AdaBoost which comes from ADAptive BOOSTing. The sampling distribution starts from uniform; as the classifiers incorrectly estimate a certain case, probability of selecting the case is increased. Algorithm will push classifier to focus on difficult cases by increasing their numbers at the sample dataset for the previous run. The boosting algorithm summarized in Figure 2, begins by assigning equal weight to all instances in the training data. The learning algorithm is called to form a classifier for this data and each data is weighted again according to the classifier's output. The weight of correctly classified instances is decreased, and that of misclassified is increased as in (2):

$$weight = \frac{weight \times e}{1 - e} \tag{2}$$

Specifically, how much weights are altered in each iterations depend on the current classifier's overall error *e*. As opposed to bagging algorithm, models do not get equal vote. Instead, in boosting, models get vote according to their error rates on the data as in (3)

$$weight = -\log\left(\frac{e}{1-e}\right) \tag{3}$$

Boosting algorithm returns the class with the highest weight.

Model generation
Assign equal weight to each training instance
For each of k iterations:
    Apply learning algorithm to weighted dataset and store resulting model
    Compute error e of model on weighted dataset and store error.
    If e equal to zero, or e greater or equal to 0.5:
        Terminate model generation.
    For each instance in dataset:
        If instance classified correctly by model:
            Multiply weight of instance by e / (1 – e).
    Normalize weight of all instances.

Classification
Assign weight of zero to all classes.
For each of the k (or less) models:
    Add –log(e / (1 – e)) to weight of class predicted by model.
Return class with highest weight.

Figure 2. Algorithm for boosting.

Data Mining in Practice

Data mining users today are in the unprecedented position of being able to collect vast amount of data relatively easily. With the advent of fast and inexpensive hardware, vendors implement software that facilitates mining of these large volumes of data.

Data mining has wide and diverse application areas ranging from business to science. Typical business applications of data mining are in the areas of e-commerce, retail and telecommunication, in marketing, the primary application is database marketing systems, which analyze customer databases to identify different customer groups and forecast their behavior. Numerous companies use data mining for investment. Data mining is widely applied in such scientific disciplines as chemistry, biology, genetics, astronomy and medicine.

There are many commercial and open source tools, techniques and products that help performing mining operation on large databases. Some of these are general purpose tools that include algorithms and techniques from various set of functionalities such as SAS, Mineset, Clementine. Some other tools are specialized in a small set of functionalities or to a specific application domain such as Hyperion (Oracle and Hyperion), Neural Connection (Neural Connection), Answer Tree (Answer Tree). This latter category of tools is developed by collecting the specific requirements of the domain and suggesting a custom solution that meets knowledge extracting needs of the users. Various standards related to data mining are defined and there are emerging data mining standards in order to manage tools and products coherently with data mining functionalities and processes.

CHAPTER III

DATA MINING STANDARDS

In a typical KDD process it is necessary to apply algorithms from different functionalities in a sequence. For instance, to solve a customer segmentation problem in marketing, a descriptive functionality implemented by a clustering algorithm is followed by a classification model. As most of the tools are generally specialized in a particular functionality of data mining during the course of a KDD project it is necessary to use different tools and these tools should communicate with each other in such a way that output of one is input to the other in the following step of KDD. Consequently, proper integration between data mining tools is very important. That kind of integrations can only be established only if proper standard interfaces are defined and implemented by these tools.

Another usage of data mining standards is in performance testing or model comparison, a user may wish to apply the same algorithm in different vendors' tools. This process necessitates the access of those tools from the same application which will be greatly simplified with the aid of a standard data mining APIs (application programming interface).

In addition, with the aid of common standards it is much more flexible and easier to extend the coverage and scope of open source tools.

In order to handle challenges of many different aspects of data mining projects, data mining standards are proposed and being used. With the emergence of standards, companies can better integrate their systems and they can use capabilities of more than one data mining vendor. That will increase the diffusion of data mining technologies into domain specific applications. For instance a CRM application may utilize various data mining algorithms implemented by different vendors.

In the recent years a couple of data mining standards have matured and data mining tools and products are utilizing these standards. The main reason why many different standards exist today is that data mining is used in so many different ways and in so many different systems that each of these often require its own standards. In spite of the fact that standardization effort about data mining functionalities has been put forward by vendors, industry needs more work to have mature standards especially related to data mining in operational processes, models and systems.

Emerging Data Mining Standards

In order to support interactive data mining and to facilitate flexible and effective knowledge discovery, data mining query language has been proposed which is challenging due to wide spectrum of mining tasks (Han and Kamber, 2005).

Today's data mining market can be compared with the database market about 40 years ago where there is no relational concept or SQL, each vendor had its own propriety storage format, and there was no easy way to query different data sources (Tang and Jamie, 2005).

One of the first standards in the data mining space is the Predictive Model Markup Language (PMML) developed by Data Mining Group (DMG) (Data Mining Group, 2007). PMML described standards of interchanging of data mining models among systems in a vendor-neutral format.

Web services are emerging as a standard mechanism for developing remote and distributed data mining applications. A recent attempt at a standardized application programming interface in the online analytical processing and business intelligence is XML for Analysis (XMLA) (XML for Analysis, 2007). XML for Analysis is designed specifically for standardizing the data access interaction between a client application and a data provider over the web allowing client applications to talk to multi-dimensional data sources.

The query language MDX (Multidimensional Expression) is the most commonly used multi-dimensional expression language today. In today's Business Intelligence marketplace, most OLAP servers and almost all BI clients talk MDX (Tang and Jamie, 2005).

The Cross Industry Standard Process for Data Mining (CRISP-DM) was a project to develop an industry- and tool-neutral data mining process model (CRISP-DM, 2007). CRISP-DM's goal is to make data mining tasks more manageable and reliable by standardizing the data mining phases, integrating and validating best practices from experts in diverse industries. In CRISP-DM standard life cycle of a data mining project consists of six phases as shown in Figure 3. The sequence of the

phases is not mandatory; moving back and forth between different phases is always needed.

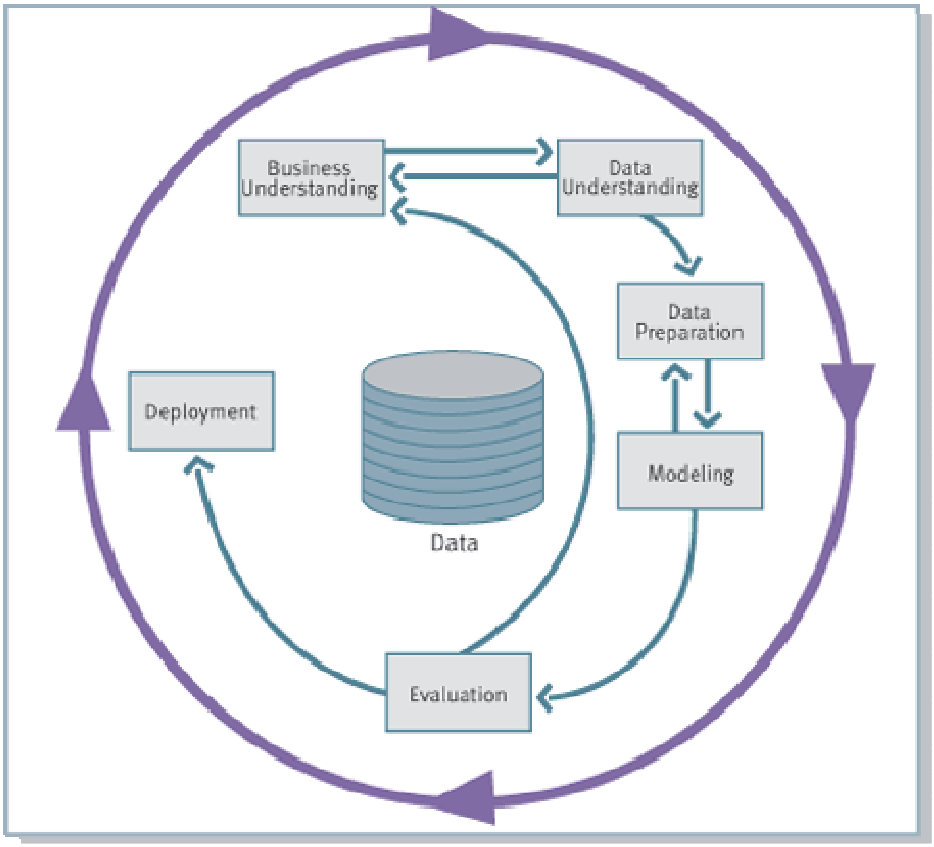

Figure 3. Phases of the CRISP-DM process model

Issues about privacy-preserving data mining (PPDM) have emerged globally. Analyzing what right to privacy means is fraught with problems, such as the exact definition of privacy, whether it constitutes a fundamental right, and whether people are and/or should be concerned with it. Several definitions of privacy can be given, and they vary according to context, culture, and environment (Oliveira and Zaiane, 2004).

Object oriented data mining standards are emerging technologies that are enabling flexible application architectures, which provide developers and businesses greater choice in how they address data mining solutions. Common Warehouse

Metadata (CWM) (Object Management Group, 2007) is a widely accepted object oriented data mining standard which provides a model for representing data mining metadata in XML. Information and data mining tasks are represented in the form of form objects that are reusable, scalable and portable.

Java Data Mining (JDM) is another well-known and well-established object oriented data mining standard.

Java Data Mining (JDM)

Java Data Mining (JDM) is a standard Java API for developing data mining applications and tools. JDM defines an object model and Java API for data mining objects and processes. JDM enables applications to integrate data mining technology for developing predictive analytics applications and tools. Traditionally, data mining algorithms were home - grown and plugged into applications without standard APIs. Therefore, the ability to embed data mining end-to-end in applications using commercial data mining products was difficult. As such, the ability to leverage data mining functionality easily via a standards based API greatly reduces the risk of selecting a particular vendor’s solution as well as increases accessibility of data mining to application developers. Java Data Mining (JDM) addresses this need.

JDM which is developed on J2EE allows users to draw on the strengths of multiple data mining vendors for solving business problems, by applying the most appropriate algorithm implementations to a given problem without having to invest resources in learning each vendor's proprietary API. J2EE is a development environment which is independent of hardware systems and operating systems. The idea behind J2EE is that software developers need only write applications once and

these applications can then be run unchanged on any computer or operating system. Java technology, specifically as leveraged within the scalable J2EE architecture, facilitates integration with existing applications such as business-to-consumer and business-to-business web sites, customer care centers, campaign management, as well as new applications supporting national security, fraud detection, bioinformatics and life sciences. Moreover, vendors and customers can focus on functionality, automation, performance, and price. With JDM's extensible framework for adding new algorithms and functionality, vendors can still differentiate themselves while providing developers with a familiar paradigm.

JDM supports various phases of CRISP-DM processes (Hornick, Marcade and Venkaya, 2007). It defines the software architecture necessary to link standardized data mining tasks with the overall data mining project's goal by ensuring the qualities of the work at each step. For instance, data mining project success depends heavily on the data available and the quality of that data; JDM supports data understanding phase through its statistics interface and allows Java applications to communicate with data mining engines to build, test and apply mining models. Whole cycle of data mining project defined at CRISP-DM is addressed at the JDM architecture.

First release of JDM published in 2004 is JDM 1.0 (JSR-73) which is based on a highly-generalized, object-oriented, data mining conceptual model leveraging emerging data mining standards such OMG's CWM, and DMG's PMML.

Data mining functionalities that are supported by JDM 1.0 are classification, regression, attribute importance, clustering, association and anomaly detection. Currently developing version JDM 2.0 (JSR-247) adds time series, transformations, feature extraction, statistics, text mining and multi-target models to the existing capabilities of JDM. On the other hand certain utilities are excluded from the scope of the current version of JDM such as visualization, mining image, mining audio, mining video, wrappers, ensembles, cross validation, bootstrapping and sensitivity analysis.

JDM further includes the specification of a web services interface based on the JDM UML model, thereby enabling Service Oriented Architecture (SOA) design. Although JDM - based web services map closely to the Java interface, JDM web services address needs beyond the Java Community, being based on Web Services Description Language (WSDL) and XML, a programming language neutral interface. Now, vendors of JDM can leverage their investment in a JDM server for both the Java and web service interfaces, using common metadata, object structure, and capabilities. However, non-JDM vendors can also leverage this same interface to be interoperable with a broader range of vendor implementations.

Data mining tools provide a wide range of options for integrating with enterprise software architectures and processes. Data mining tools can be viewed as consisting four parts: a graphical user interface (GUI) for interactive data mining, an application programming interface (API) for building applications, a data mining engine (DME) where the core processing or data mining algorithm execution occurs, and a mining object repository (MOR) where persistent mining objects are stored as

shown in Figure 4. Data mining tools are further categorized along such dimension as data access option, mining object storage, and DME location. JDM architecture as illustrated in Figure 5. accesses data from flat files or databases, MOR is located in a database, and DME objects run independent with MOR inside JDM.

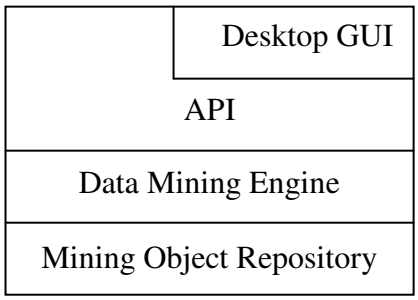

Figure 4. Data mining tool architecture

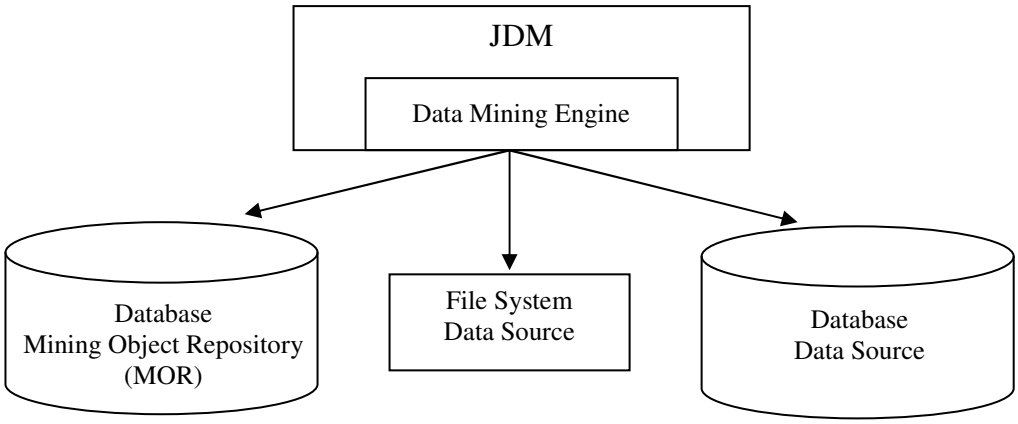

Figure 5. JDM architecture

The JDM model supports conceptual areas that are generally of key interest to users of data mining systems: settings (BuildSettings, ApplySettings), models, transformations, and results. The object model provides a core layer of services and interfaces that are available to all clients. Object modeling of the JDM API begins with *MiningObject* where all other objects are derived from at the JDM as shown in Figure 6. *MiningObject* is a serializable and comparable object than can be saved with a name to a mining object repository (MOR) via data mining engine (DME).

*MiningObject*s in JDM are *BuildSettings*, *Model, Task, TestMetrics, LogicalData, PhysicalDataSet* and *ApplySettings.*

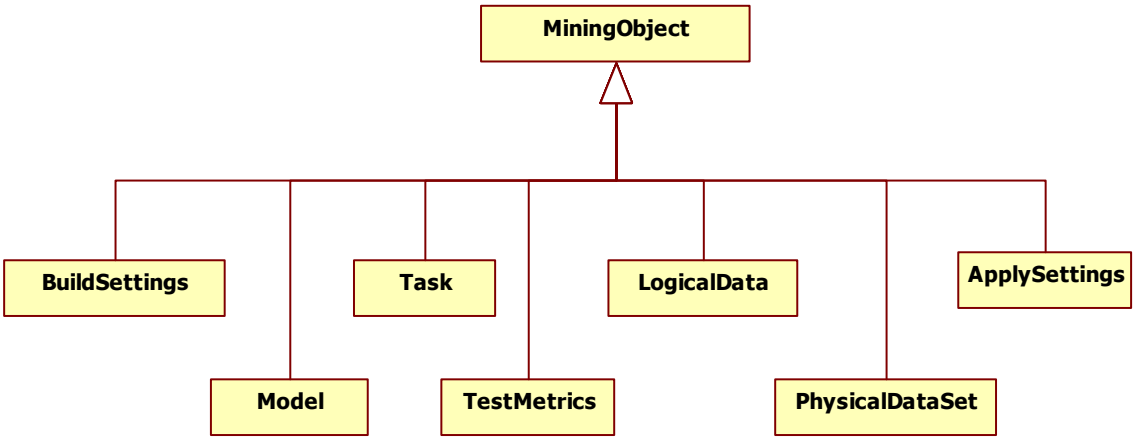

Figure 6. JDM mining objects

PhysicalDataSet and *LogicalData* are data specification objects that are used to describe the input data for mining. By using these objects DME know data attribute characteristics so data can be mined effectively.

BuildSettings and *ApplySettings* are settings objects; settings objects involve specification of various settings. To build a model, users specify settings at mining function level and optionally at the algorithm function level. For example, a classification function can use several algorithms such as decision tree, naïve bayes, neural networks or support vector machine.

A *model* object contains a compact representation of the knowledge. In JDM, each mining function has an associated model object such as ClassificationModel, ClusteringModel. A model may also contain algorithm-specific representation, for example TreeModelDetail provides the model details specific to the decision tree algorithm in classification functionality.

TestMetrics object provide insight into how a model performs relative to test data, as well as how a model performs in comparison with other models.

Task objects support the JDM operations like build, test, apply, compute statistics, as well as import and export by providing synchronous and asynchronous execution option through data mining engine.

JDM uses a vendor-neutral connector architecture that allows applications to connect a data mining engine with interoperable code. Using the Java Naming and Directory Interface (JNDI) and factory method pattern, JDM isolates the vendor specific connection implementation. All objects of JDM are defined as interfaces. In order to provide an implementation, these interfaces should be implemented by a vendor. Object instantiation can only be done by invoking the *getFactory()* method of the connection object which serves as an abstract factory for factories or by invoking the create methods of the objects that are driven from factories.

As a part of solution of a clustering functionality, Figure 7 illustrates the instantiation of *ClusteringSettings* object from *ClusteringFactory*'s create method. *ClusteringSettings* inherits *BuildSettings MiningObject* base class. There are enumeration type classes namely *AttributeComparisonFunction* and *AggregationFunction*. Enumaration type classes in JDM are collections of options that are manually or automatically set during execution. In order to build a clustering model *ClusteringSettings* object is created. Using *setAggregationFunction()* method one of the aggregation functions in the enumeration class is selected and using

setAttributeComparionFunction() method one of the attribute comparison functions is selected.

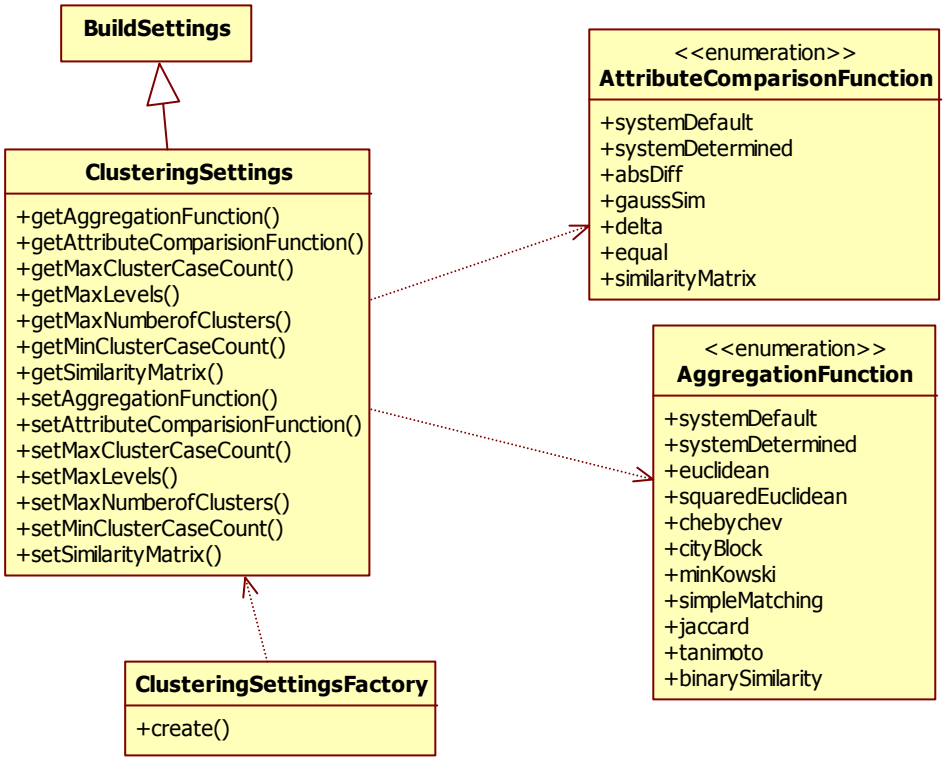

Figure 7. *ClusteringSettings* object

Building a clustering model is illustrated in Figure 8., assuming that dataset is stored physically at *myBuildData* table in the given uniform resource identifier (URI), *PhysicalDataset* object is created at lines between 1 and 3. *ClusteringSettingsFactory* object and *ClusteringSettings* object are created at line 4 and line 5. Max number of cluster is set to 20 and minimum cluster case count is set to 5 at lines 6 and 7. Clustering setting is saved to mining object repository named as *myClusteringBS* at line 8. *BuildTask* object is created and is sent to the DME for execution at lines between 9 and 12.

```
1. PhysicalDatasetFactory pdsFactory = (PhysicalDataSetFactory)
       dmeConn.getFactory("javax.datamining.data.PhysicalDataSet");
2. PhysicalDataset buildData = pdsFactory.create(uri,true);
3. dmeConn.saveObject("myBuildData",buildData,false);
4. ClusteringSettingsFactory csFactory = (ClusteringSettingsFactory)
       dmeConn.getFactory("javax.datamining.clustering.ClusteringSettings");
5. ClusteringSettings clusteringSettings = csFactory.create();
6. clusteringSettings.setMaxNumberOfClusters(20);
7. clusteringSettings.setMinClusterCaseCount(5);
8. dmeConn.saveObject("myClusteringBS",clusteringSettings);
9. BuildTaskFactory btFactory = (BuildTaskFactory)
       dmeConn.getFactory("javax.datamining.task.BuildTask");
10. BuildTask task =
       btFactory.create("myBuildData","myClusteringBS",
       "myClusteringModel");
11. dmeConn.saveObject("myClusteringTask",task);
12. ExecutionHandle handle = dmeConn.execute("myClusteringTask");
```

Figure 8. Code segment of building a clustering model.

As with any standard, defining conformance for vendor implementations raises myriad issues. Implementations are not necessarily support all algorithms and features. In JDM, compliance is based on a core feature set with optional packages for each mining function and algorithm. In addition, JDM provides *supportsCapability()* methods that allow applications to determine at runtime if a particular vendor implementation supports a finer grained feature, e.g., whether classification model build accepts a cost matrix specification, or the clustering algorithm produces hierarchically arranged clusters. For those features the vendor supports of a valid JDM configuration, the vendor implementation must pass the TCK which is Technology Compatibility Kit for vendors.

Vendor implementations are key to the success of any standard. Currently Oracle and KXEN are the two primary vendors that implemented JDM 1.0 (Oracle Data Mining, 2007), (Kxen Java Data Mining, 2007). Vendors implement JDM interfaces either by coding classes and methods defined by JDM or by mapping JDM interfaces to the existing data mining objects that are already provided by vendors. KXEN joined the JDM standards group because the concept that were beginning to emerge in this vendor in 2002 were close to the architecture decisions KXEN made in 1998, when designing the KXEN Analytic Framework. To extend to our knowledge, KXEN JDM supports all JDM basic mining functions. Oracle is a strong supporter of open standards developed under Java Community Process (JCP). The JDM 1.0 standard was initiated and led by Oracle under the JCP. Proposed design and applications are implemented using Oracle JDM 1.0.

CHAPTER IV

DESIGN AND IMPLEMENTATIONS

Although, assessing the accuracy of classification models is an essential task of data mining studies, this task is not included in the currently developing version of JDM. We suggested bootstrap sampling method based on the class hierarchy of JDM. In addition, improving the accuracy of classification models by combining different classifiers is important in utilization of these models; therefore we also suggested bagging and boosting as part of JDM standard.

In this study, we are using Oracle JDM infrastructure to implement our suggested experimental framework. Oracle JDM maps the JDM named objects, such as mining model, task, settings objects, and test metrics to its database objects. However these implementations are not open source. As we develop our experimental framework for academic purposes, we have to use an emulator-based architecture. In order to implement our suggested JDM interfaces, original class diagrams are extended with emulator classes to provide a gateway to Oracle JDM. For instance, *Bootstrap* class as shown in Figure 9. is an emulator class which provides a gateway to Oracle JDM for applying bootstrap technique.

Bootstrap Structure

Using JDM interfaces, classes, and standards, *Bootstrap* object can be created using *BootsrapFactory* object. Figure 9 depicts bootstrap related classes and their relationships. User can specify dataset, number of iterations (k) and a predictive algorithm (buildsettings) using the appropriate functions of *BootstrapSettings* class.

getAccuracy() method is the most essential function that comprise bootstrap algorithm as shown in Figure 15. *BootstrapErrorMeasure* defines the enumerated values that can be referenced as error measures that are used for calculation of regression models. Bootstrap class is used as a gateway to Oracle JDM implementation by invoking JDM database objects iteratively as needed for bootstrap algorithm. Figure 10, Figure 11 and Figure 12 show methods of *BootstrapSetting*, methods of *Bootstrap* class and methods of *SamplingwithReplacement* respectively.

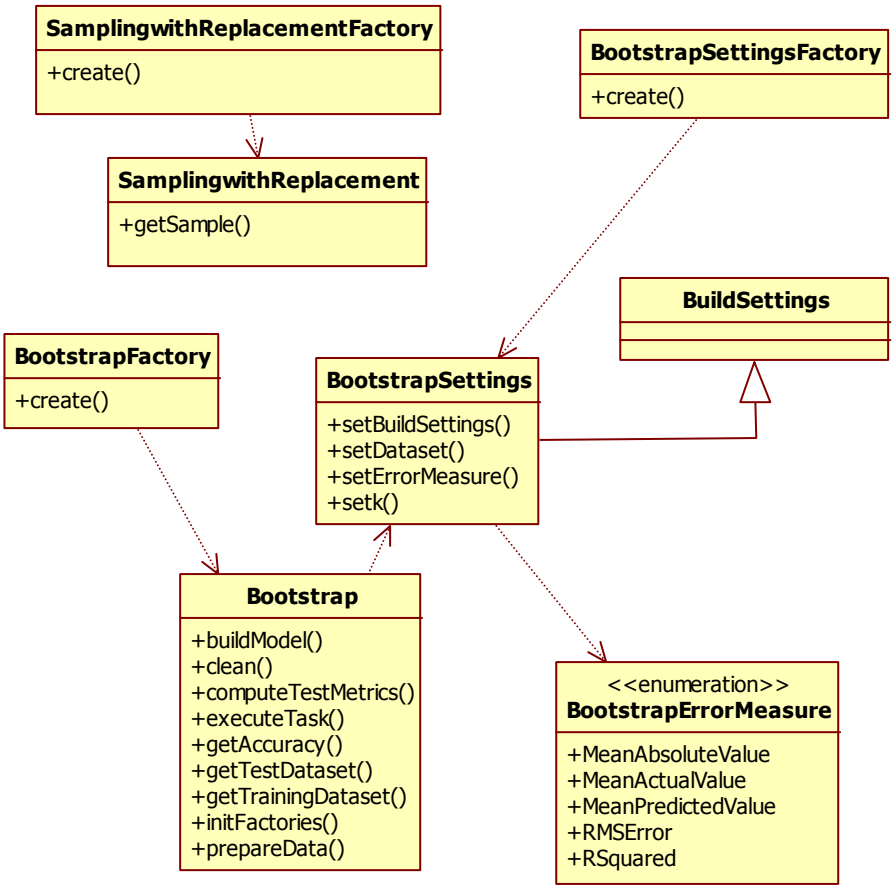

Figure 9. Bootstrap class structure used in JDM

The BootstrapSettingsFactory Class

BootstrapSettings create()
This method creates an instance of *Bootstrapsettings* initialized to vendor-specific default values.

The BootstrapSettings Class

setBuildSettings(BuildSettings)
This method sets Buildsetting which comprises the functionality and its paramaters. The method receives Buildsettings object as its parameter.

setDataset(physicaldataset)
This method sets dataset before later steps of Bootstraping. The method receives physical dataset as its parameter.

setErrorMeasure(BootstrapErrorMeasure)
This method defines the type of accuracy calculation. If not defined, mining engine can automatically assign a default type of accuracy calculation for regression models. The method receives the optional parameter of BootstrapErrormeasure.

setk(number of iterations)
This method sets number of iterations indicating how many times the bootstrap procedure will be performed. It receives number of iterations as its parameter.

Figure 10. Methods of the *BootstrapSettings* and *BootstrapSettingsFactory* classes

The BootstrapFactory Class

Bootstrap create()
This method creates an instance of Bootstrap initialized to vendor-specific default values.

The Bootstrap Class

buildModel(physicaldataset)
This method uses BuildModel command object specified at JDM 2.0. The model is built using sampled training dataset. Method receives physical dataset as parameter.

clean()
This method cleans previous JDM objects that are used during execution of the code.

computeTestMetrics(applyOutputName, Datasetname)
This method uses previously trained model name and a solicited dataset name, this method computes accuracy of the model on the dataset. The method receives apply output name and dataset name as parameters.

executeTask(Taskobj, taskname)
This method starts and handles execution request of the *Task* object. The method returns true if execution process is successful or returns false if execution process is failed. The method receives task object and task name as its parameters.

getAccuracy()
This method returns the accuracy of the model given dataset and the model. This method uses the k number specified at the BootstrapSettings class and iterates k times in order to calculate overall accuracy as shown in Figure 15.

getTestDataset()
This method returns the test dataset. The instances that are not selected for training purpose are treated as test instance.

getTrainingDataset()
This method returns the sampled training dataset.

initFactories()
This method initiates factory objects that are used by the Bootstrap objects.

prepareData()
This method uses SamplingwithReplacement class and methods and prepares a new sampled training and test datasets.

Figure 11. Methods of the *Bootstrap* and *BootstrapFactory* classes

The SamplingwithReplacementFactory Class

Samplingwithreplacement create()
This method creates an instance of SamplingwithReplacement initialized to vendor-specific default values.

The SamplingwithReplacement Class

getSample(physicaldataset)
This method performs sampling with replacement on a given dataset and returns the sampled dataset as of type physicaldataset. The method receives physical dataset as its parameter.

Figure 12. Methods of the *SamplingwithReplacement* and *SamplingwithReplacementFactory* classes

Sampling with Replacement

Sampling with replacement algorithm is the core of Bootstrap algorithm. Given JDM connection object and table name (the variable str_dataset), the executing code copies data to a temporary table and extends the structure by adding a new column that contains an artificial ID. Figure 13 shows how SQL string is prepared. Due to architectural design of JDM standard, data mining engine backing vendor implementations is a relational database. An important logic behind relational databases is usage of keys. Primary keys are enforced to be unique in a table; however sampling with replacement algorithm may select a record more than once.

```
String cpysql = "CREATE TABLE " + cpy_dataset +  " (A_ID NUMBER NOT NULL";
```

Figure 13. Creating the artificial ID

After creating the temporary table, random numbers are generated. The number of records in the table, *casenum* is used to generate integer numbers between ranges 1 to *casenum* stored in an array. Figure 14 shows a working Java code example, an integer array of size *casenum* and generated integers recorded to the array.

```
1. int [] a = new int [casenum];
2. Random randomGenerator = new Random();
3. for(int j=0;j<a.length;j++){
4. a[j] = 1 + randomGenerator.nextInt(casenum); }
```

Figure 14. Code segment of random selection

Generated integer numbers are used while building training dataset where there are almost certainly duplicated instances. Unselected integers are used as test dataset. For a table with 10 records (casenum is 10), suppose randomly generated integers have values of {1,1,3,4,5,5,6,6,6,8}. Records of 1,3,4,5,6 and 8 will be used as training dataset where 1 and 5 will appear twice and 6 will exist 3 times at the training set. Test set will consist of the records that are not included in the training dataset, these are {2,7,9,10}.

Accuracy

Set Overall Accuracy "A" to zero.
For each of k iterations
    Invoke sample() method
    Build a model with sampled Training Dataset and predefined BuildSettings.
    Calculate accuracy of Training Dataset on the built model
    Calculate accuracy of Test Dataset on the built model
    Combine the accuracy values of Training and Test Datasets as A = A +
(Accuracy of Test Dataset x 0.632 + Accuracy of Training Dataset x 0.368)

Return A/k as the overall accuracy of the model on the given dataset.

Figure 15. Algorithm for *getAccuracy()* method

Bagging Structure

Proposed *Bagging* object can be instantiated by using *BaggingFactory* object. Figure 16 illustrates classes for *Bagging* and their relationships. User can specify dataset, number of iterations (k) and a predictive algorithm (buildsettings) using the appropriate functions of *BaggingSettings* class. The *getAppliedSetting()* method is the most essential function that comprises bagging algorithm as shown in Figure 22. Similar to bootstrap *Bagging* class is used as a gateway to Oracle JDM implementation by invoking JDM database objects iteratively as needed for bagging algorithm. Figure 19, Figure 20 and Figure 21 show methods of BaggingSettings, Methods of Bagging class and Methods of *SamplingwithReplacement* respectively.

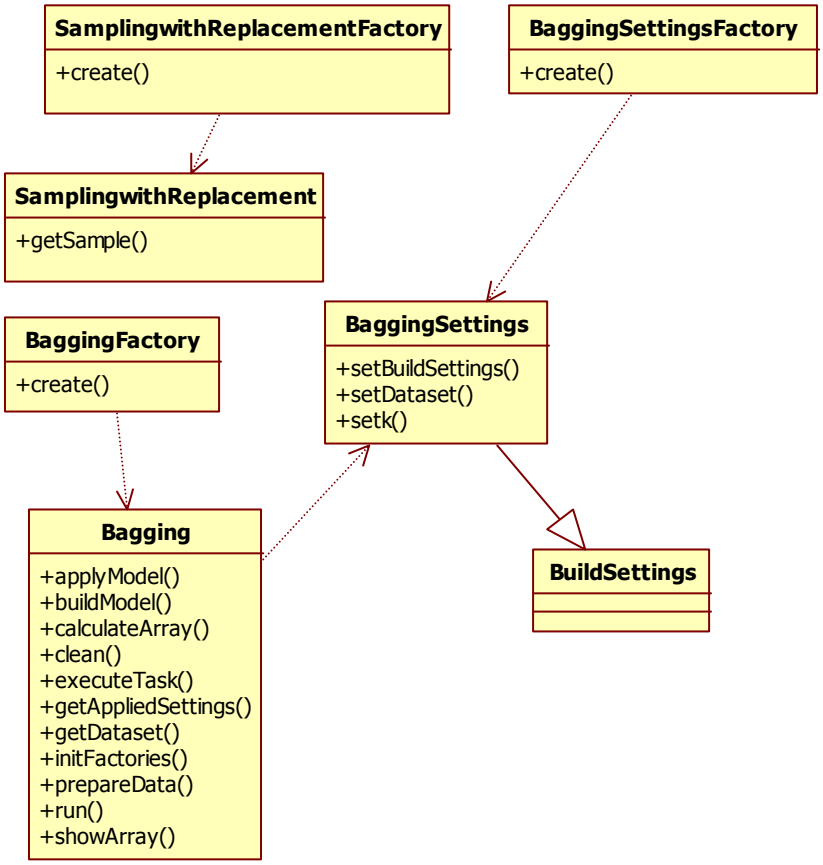

Figure 16. Bagging structure used in JDM

In order to create Bagging object and run algorithm, A *BaggingSettings* object should be instantiated as shown in Figure 17. In addition, dataset names, number of k and *buildSettings* are defined at lines between 3 and 5.

```
1. BaggingSettingsFactory bsf = new BaggingSettingsFactory();
2. BaggingSettings bs = bsf.create();
3. bs.setBuildSettings(buildSettings);
4. bs.setk(100);
5. bs.setDataset("SRC_DATA","CPY_DATA","TRN_DATA ","TST_DATA")
```

Figure 17. Code segment of instantiation of *BaggingSettingsFactory* class

Bagging object *bagg* can only be instantiated from *BaggingFactory* and by using *BaggingSettings* object as its parameter to create method of factory object as shown in Figure 18.

```
1. BaggingFactory bf = new BaggingFactory();
2. Bagging bagg = bf.create(bs);
3. bagg.run(m_dmeConn);
```

Figure 18. Code segment of instantiation of *Bagging* class

The BaggingSettingsFactory Class

BaggingSettings create()
This method creates an instance of *BaggingSettings* initialized to vendor-specific default values.

The BaggingSettings Class

setBuildSettings(BuildSettings)
This method sets *BuildSetting* which comprises the functionality and its paramaters. The method receives *BuildSettings* as its parameter.

setDataset(physicaldataset)
This method defines before later steps of Bagging. Method receives physical dataset as its parameter.

setk(number of iterations)
This method sets number of iterations indicating how many times the bagging procedure will be performed. The method receives number of iterations as its parameter.

Figure 19. Methods of *BaggingSettings* and *BagginSettingsFactory* classes

The BaggingFactory Class

Bagging create()
This method creates an instance of Bagging initialized to vendor-specific default values.

The Bagging Class

applyModel(physicaldataset)
This method uses ApplyModel command object specified at JDM 2.0. Model is built using sampled training dataset. The method receives physical dataset as its parameter.

buildModel(physicaldataset)
This method uses BuildModel command object specified at JDM 2.0. Model is built using sampled training dataset. The method receives physical dataset as its parameter.

calculateArray(physicaldataset)
This method calculates array of dataset based on equal votes from each classifier. The classifier that receives the highest votes is predicted as the result of the bagging model. The method receives physical dataset as its parameter.

clean()
This method cleans previous JDM objects that are used during execution of the code.

executeTask(Taskobj, taskname)
This method starts and handles execution request of the Task object received as one of its parameters. The methods returns true if execution process is successful, returns false if execution process is failed. The method receives task object and task name as its parameters.

getAppliedSetting(physicaldataset)
This method takes an array of dataset and returns the predicted values as an array of dataset based on the algorithm shown in Figure 22. The method receives physical dataset as its parameter.

getDataset(DatasetName)
This method applies bagging models on dataset specified at its parameter and calculates the votes of the all individual records at the dataset. The method receives dataset name as its parameter.

run()
This method run k models that are trained and saved at run-time using specified parameters and sampled datasets.

showArray()
This method reports the predicted class of the records by using the majority votes. It is invoked at the last step of *getdataset()* method.

Figure 20. Methods of the *Bagging* and *BaggingFactory* classes

The SamplingwithReplacementFactory Class

Samplingwithreplacement create()
This method creates an instance of SamplingwithReplacement initialized to vendor-specific default values.

The SamplingwithReplacement Class

getSample(physicaldataset)
This method performs sampling with replacement on a given dataset and returns the sampled dataset as of type physicaldataset. Method receives physical dataset as parameter.

Figure 21. Methods of *SamplingwithReplacement* and

SamplingwithReplacementFactory classes

Bagging

Initialize Array of results

For each of k iterations
    Invoke sample() method
    Build a model with sampled Training Dataset and predefined BuildSettings.
    Apply model to the data set
    Update counts of predicted results at the Array

Calculate Array of results
Return the Array of results.

Figure 22. Algorithm for *getAppliedSetting()* method.

Boosting Structure

In JDM architecture, *Boosting* object is created using *BoostingFactory* object. Figure 23 illustrates classes for *Boosting* and their relationships. User can specify dataset, number of iterations (k) and a predictive algorithm (buildsettings) using the appropriate functions of *BoostingSettings* class. Similar to bootstrap and bagging *Boosting* class is used as a gateway to Oracle JDM implementation by invoking JDM database objects iteratively as needed for boosting algorithm. Figure 24, Figure 25

and Figure 26 show Methods of BoostingSettings, Methods of Boosting class and Algorithm of run() method respectively.

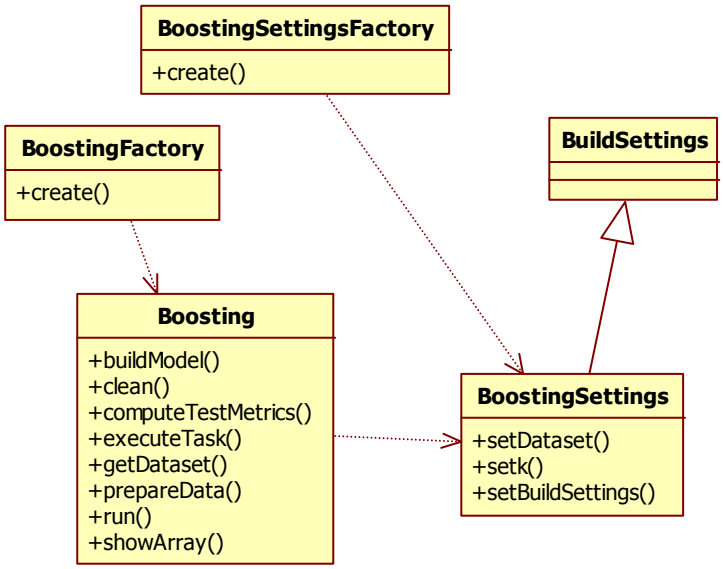

Figure 23. Boosting structure used in JDM

The BoostingSettingsFactory Class

BoostingSettings create()
This method creates an instance of *Boostingsettings* initialized to vendor-specific default values.

The BoostingSettings Class

setBuildSettings(BuildSettings)
This method sets *Buildsetting* which comprises the functionality and functionality parameters. The method receives build setting as its parameter.

setDataset(physicaldataset)
This method defines the dataset before later steps of Boosting. The method receives physical dataset as its parameter.

setk(number of iterations)
This method sets number of iterations indicating how many times the Boosting procedure will be performed on the given data is set. The method receives physical dataset as its parameter.

Figure 24. Methods of the *BoostingSettings* and *BoostingSettingsFactory* classes

The BoostingFactory Class

Boosting create()
This method creates an instance of Boosting initialized to vendor-specific default values.

The Boosting Class

buildModel(physicaldataset)
This method uses *BuildModel* command object specified at JDM 2.0. Model is built using sampled training dataset. The method receives physical dataset as its parameter.

clean()
This method cleans previous JDM objects that are used during execution of the code.

computeTestMetrics(applyOutputName, Datasetname)
This method uses previously trained model name and a solicited dataset name, this method computes accuracy of the model on the dataset. The method receives apply output name and dataset name as parameters.

executeTask(Taskobj, taskname)
This method starts and handles execution request of the Task object received as one of its parameters. The method returns true if execution process is successful, returns false if execution process is failed. The method receives task object and task name as its parameter.

getDataset(DatasetName)
This method applies Boosting models on dataset specified at its parameter and calculates the votes of the all individual records at the dataset.

prepareData()
This method uses *SamplingwithReplacement* class and methods, a new sampled training and test datasets are prepared.

run()
This method trains k models and saves models at run-time using specified parameters and sampled datasets.

showArray()
This method reports the predicted class of the records by using the majority votes. This method is invoked at the last step of *getdataset()* method.

Figure 25. Methods of the *Boosting* and *BoostingFactory* classes

Boosting

Initialize Array of results

For each of k iterations
    Invoke sample() method
    Build a model with sampled Training Dataset and predefined BuildSettings.
    Apply model to the data set
    Update counts of predicted results at the Array using error rate

Calculate Array of results
Return the Array of results.

Figure 26. *run()* method of boosting algorithm

Error rates of models are stored in array named Error for later steps of boosting algorithm as shown in Figure 27. Accuracy of model on training dataset and test dataset can be retrieved using *computeTestMetrics* method and *ModelAccuracy* is calculated. 1 – *ModelAccuracy* is stored to the array for later usages. An essential piece of code is highlighted however full implementation code is given in appendix.

```
1. a1=computeTestMetrics("DT_TRN_APPLY_OUTPUT_JDM" +
   i,"dtTestMetrics_jdm", null, str_sample,i);
2. a2=computeTestMetrics("DT_TEST_APPLY_OUTPUT_JDM" +
   i,"dtTestMetrics_jdm", null, tst_dataset,i);
3. ModelAccuracy = a2 * 0.632 + a1 * 0.368;
4. System.out.println(ModelAccuracy);
5. Error[i] = 1 - ModelAccuracy;
```

Figure 27. Code segment of combining accuracy values.

CHAPTER V

APPLICATION.

Three data mining techniques, bootstrap, bagging, and boosting over JDM architecture were proposed and these design and implementations were applied to datasets in order to ensure that they are working as they are suggested, stable enough to run for real datasets. Bootstrap, bagging and boosting procedures were applied to two datasets. One is a well – known dataset in the literature the iris dataset (Asunction and Newman, 2007) and the other is a well - known dataset in the Oracle world The SH (Sales History) dataset (Oracle Data Mining, 2007).

Iris Dataset

The iris dataset is a popular multivariate dataset that was introduced by R.A. Fisher as an example for discriminate analysis. There are 3 types of iris plant and type of the plant is predicted using other attributes in the dataset. Number of instances is 150, predicted variable is the class of iris plant. Predictive attributes are shown Table 1.

Table 1. Predictive attributes

sepal length in cm
sepal width in cm
petal length in cm
petal width in cm

Distinct values of the predicted attribute are iris setosa, iris versicolour or iris virginica. There are 50 instances for each class in the dataset. There are no missing attribute values. Table 2. shows summary statistics of predictive attributes. SL refers

to sepal length of the plant. SW refers to sepal width of the plant. PL refers to petal length of the plant. PW refers to petal width of the plant.

Table 2. Summary statistics.

	Min	Max	Mean	Std
SL	4.30	7.90	5.84	0.83
SW	2.00	4.40	3.05	0.43
PL	1.00	6.90	3.76	1.76
PW	0.10	2.50	1.20	0.76

Numeric attributes are partitioned as small, medium and large using equiwidth binning technique. Table 3 shows lower and upper boundaries of SL, SW, PL and PW.

Table 3. Binned statistics.

	Small	Medium	Large
SL	4.30 - 5.50	5.50 - 6.70	6.70 - 7.90
SW	2.00 - 2.80	2.80 - 3.60	3.60 - 4.40
PL	1.00 -2.97	2.97 - 4.93	4.93 - 6.90
PW	0.10 -0.90	0.90 - 1.70	1.70 -2.50

Table 4 shows number of instances in each category. There are 50 small plants in term of their PW and PL. 99 out of 150 instances are medium based on their sepal width. There are only 18 large plants in terms of their SW but there are 70 nearly half of them is medium in terms of their SL.

Table 4. Number of instances in small, medium and large categories.

	S	M	L
SL	52	70	28
SW	33	99	18
PL	50	54	46
PW	50	52	48

Table 5 shows the distribution of setosa type plant in small, medium and large categories. It is noticeable that there are 0 (zero) instances in some categories for

instance large SL, medium and large PL, medium and large PW. There is only 1 instance with small sepal width. Based on this table, basic inference when the instance is setosa is that there is very little or no chance of being medium or large category in terms of petal length or petal width, being small category in terms of sepal width or being large category in terms of sepal length.

Table 5. Number of setosa instances in small, medium and large categories.

setosa	S	M	L
SL	45	5	0
SW	1	34	15
PL	50	0	0
PW	50	0	0

Table 6 shows the distribution of versicolor instances in small, medium and large categories. There are no instances at large SW, small PL and small PW categories. There are only 2 instances at large PL and large PW categories. Versicolor iris plants are possibly medium size in terms of their petal length and their petal width.

Table 6. Number of versicolor instances in small, medium and large categories.

versicolor	S	M	L
SL	6	38	6
SW	21	29	0
PL	0	48	2
PW	0	48	2

Table 7 shows the distribution of virginia instances in small, medium and large. 46 out of 50 virginia instances are categorized as large and 44 out of 50 virginia instances are categorized as large also. There is very little possibility that sepal length of a virginia iris is small in terms of sepal length, because only 1 instance is categorized as small size in terms of its sepal length.

Table 7. Number of virginia instances in small, medium and large categories.

virginia	S	M	L
SL	1	27	22
SW	11	36	3
PL	0	6	44
PW	0	4	46

Decision tree algorithm is applied to iris dataset. Decision tree algorithm in the Oracle data mining engine predicted iris data with an accuracy of 0.8333. Bootstrap procedures applied to iris dataset using suggested JDM architecture. Classification functionality and decision tree algorithm was set as *Buildsettings* and number of iterations k is set to 100. Bootstrap procedures using sampling with replacement calculated overall accuracy of model on iris dataset as 0.9174 as shown in Table 8.

Table 8. Accuracy and error rate of algorithms on iris dataset.

	Decision Tree	Bootstrap
Accuracy	0.8333	0.9174
Error Rate	0.1677	0.0826

In order to increase accuracy of estimators, bagging and boosting procedures were applied to iris dataset using suggested JDM architecture. The dataset was split into two pieces; one is 120 randomly selected instances for training dataset and 30 instances in test dataset for calculating accuracy of the models. Stratified random sampling is performed in order to keep target attribute percentages equal at training and test set (Hunt, 2004). For example, 40 setosa iris plants were randomly selected as training and 10 unselected setosa instances stored as test dataset. Bagging and boosting algorithms applied to the training dataset and models were generated to vote for the test instances. Splitting dataset into training and test sets were performed 5 times so every 30 instances in the iris data has been selected as test dataset and forecasted via rest of the training dataset's model. Accuracy of the bagging algorithm

is 0.8199 on iris dataset. On the other hand, boosting algorithm where models have contributions based on their error rates performed with an average accuracy of 0.8533 as shown in Table 9.

Table 9. Bagging vs. boosting on iris dataset

Accuracy	Bagging	Boosting
Iteration 1.	0.8333	0.8666
Iteration 2.	0.7666	0.8000
Iteration 3.	0.8000	0.8666
Iteration 4.	0.8666	0.8333
Iteration 5.	0.8333	0.9000
Average Accuracy	**0.8199**	**0.8533**
Standard Deviation	0.0380	0.03799

Assuming that accuracy values of iterations have an approximate normal distribution, using 5% significance level, t test applied whether there is a significant difference between the accuracy values of bagging and boosting methods. We can conclude that statistically the accuracy of bagging algorithm is not different from the accuracy of boosting algorithm (Mann, 1995). Critical value of the t with 8 degrees of freedom and the specified significance level is 1.86 where calculated t value is 0.88.

Sales History Dataset

SH (Sales History) database is a well - known Oracle sample dataset (Oracle Data Mining, 2007). There are 45000 customers in this table. Table 10. shows affinity card information of the customers. Having an affinity card is the target variable in this customer dataset. Affinity card variable can be 1 meaning that customer has an affinity card. Affinity card variable can be 0 meaning that customer does not have an affinity card. Target attribute is predicted using such classifiers as age, gender, customer income level, and occupation.

Table 10. Affinity card summary statistics.

AFFINITY_CARD	Count	Percentage
1	1072	24%
0	3428	76%

We applied our suggested bootstrap algorithm to the customer dataset. Decision tree algorithm in the Oracle data mining engine predicted accuracy as 0.7416 whereas bootstrap algorithm predicted accuracy as 0.7890 as shown in Table 11.

Table 11. Accuracy and error rate of algorithms on SH dataset.

	Decision Tree	Bootstrap
Accuracy	0.7416	0.7890
Error Rate	0.2584	0.2110

Table 12. shows accuracy values of boosting and bagging at each iteration. Average accuracy of bagging algorithm on SH data is 0.7187 with a standard deviation of 0.0391 whereas average accuracy of boosting algorithm on SH data is 0.7340 with a standard deviation of 0.0416.

Table 12. Bagging vs. boosting on SH dataset

Accuracy	Bagging	Boosting
Iteration 1.	0.7345	0.7710
Iteration 2.	0.6516	0.6640
Iteration 3.	0.6960	0.7343
Iteration 4.	0.7412	0.7450
Iteration 5.	0.7416	0.7560
Average Accuracy	**0.7187**	**0.7340**
Standard Deviation	0.0391	0.0416

Assuming that accuracy values of iterations have an approximate normal distribution, using 5% significance level, t test applied whether there is a significant difference between the accuracy values of bagging and boosting methods. We can conclude that statistically the mean accuracy of bagging algorithm is not different from the mean

accuracy of boosting algorithm. Critical value of the t with 8 degrees of freedom and the specified significance level is 1.86 where calculated t value is 0.38.

Oracle Enterprise Edition v.10gR2 with data mining option was used on a computer that has 1400 Mhz. CPU and 768 MB. RAM. Microsoft Windows XP used as the operating system. Version of Java Runtime was 1.6.0_03. Bootstrap iterations with k is equal to 100 last about 90 minutes. Bagging and boosting iterations with k is equal to 30 last about 45 minutes.

CHAPTER VI

CONCLUSION

Parallel to the widespread diffusion of data mining concepts and techniques in academy and industry, plenty of data mining products have emerged at the market as a part of business intelligence solutions. Without widely-accepted standards, advantages of data mining solutions cannot be fully utilized. Hence, many data mining standards have been developed. Conforming to data mining standards reduces costs and completion time of data mining projects. Data mining tools and technologies developed based on common standards facilitate diffusion of these technologies into the domain specific applications such as customer relationship management, geographical information systems, and fraud and crime detection systems.

As one of the object oriented data mining standards, JDM enables flexible application architectures for data mining solutions. In this study, we presented an experimental framework to extend the scope of currently developing version of JDM (JDM 2.0). Although, assessing the accuracy of classification models is essential part of data mining studies, currently it is not included in JDM. We suggested bootstrap sampling method to be included in JDM standard. In addition, improving the accuracy of classification models by combining different classifiers is important in utilization of these models; therefore we also suggested bagging and boosting as part of JDM standard. Due to the fact that it is not possible to interfere with Oracle's data mining engine implementations, Emulator-based architecture was suggested in

order to use JDM objects currently proposed on the top of Oracle Data Mining objects. We have applied our suggested extensions to JDM standard to two well known datasets which are iris dataset and SH dataset. We have found that in our datasets, the accuracy value of bagging algorithm is indifferent than the accuracy value of boosting algorithm. As a further study, other functionalities of data mining that are not covered yet by JDM such as sequential pattern, graph mining, spatial mining might be added to the standard.

APPENDIX

Appendix A.

CD is attached to this document. There is src folder in the CD where source files of all implementations can be found. In order to run these code Oracle Enterprise Edition 10gR2 with data mining option and Java Runtime Environment 1.6.0_03 is required.

REFERENCES

Alpaydın Ethem (2004). Introduction to Machine Learning, 332 - 333, Massachusetts Institute of Technology.

Answer Tree, http://www.spss.com/answertree/

Apoteker Thierry, Sylvain Barthelemy (2005). Predicting Financial Crises in Emerging Markets using a Composite Non-Parametric Data Mining Model.

Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, School of Information and Computer Science.

Brachman, R.J. & Anand T. (1996). Advances in knowledge discovery and data mining. CA: American Association for Artificial Intelligence.

Breiman L.(1996). Bagging Predictors, Machine Learning, 123-140

Cross Industry Standard Process for Data Mining, http://www.crisp-dm.org/

Data Mining Group, http://www.dmg.org

Efron B., R.Tibshhirani (1993). An Introduction to Bootstrap, Chapman & Hall.

Ellen Monk, Bret Wagner (2006). Concepts in Enterprise Resource Planning, Second Edition. Thomson Course Technology, Boston, MA

Fayyad U.M., S.G.Djorgovski, N. Weir (1996). Advances in Knowledge Discovery and Data Mining, MIT Press.

Freund Y., R.E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 199-139

Han J. & Kamber (2005). Data Mining: Concepts and Techniques

Han J. & Chen M., Yu P.S. Data Mining: An Overview from Database Perspective. Canada: Natural Sciences and Engineering Research Council of Canada.

Hornick Mark, Erik Marcade, Sunil Venkaya (2007). Java Data Mining Strategy, Standard and Practice, Morgan Kaufman.

Hunt Neville & Sidney Tyrrell (2004). Stratified Sampling, Coventry University, http://www.coventry.ac.uk/ec/~nhunt/meths/strati.html

Inoue Atsushi, Luts Kilian (2007). How Useful is Bagging in Forecasting Economic Time Series? A Case Study of U.S. CPI Inflation.

Java Specification Request 73: Java Data Mining (JDM) Release 1.0, http://jcp.org/en/jsr/detail?id=73

Java Specification Request 247: Java Data Mining (JDM) Release 2.0, http://jcp.org/en/jsr/detail?id=247

Kroenke M. David (2006), Database Processing, Prentice Hall, United States of America.

Kxen Java Data Mining
http://www.kxen.com/products/analytic_framework/kjdm.php

Mann Prem S.(1995). Statistics for Business and Economics. John Wiley & Sons, inc. New York: Eastern Connecticut State University.

Neural Connection 2.0,
http://www.cis.hut.fi/projects/somtoolbox/links/neural_connection.shtml

Object Management Group, http://www.omg.org/

Oliveira S.R.M & Zaiane O.R. Toward Standardization in Privacy-Preserving Data Mining. http://www.cs.ualberta.ca/~oliveira/psdm/psdm/index.html

Oracle Data Mining, http://www.oracle.com/technology/products/bi/odm/index.html

Oracle and Hyperion, http://www.oracle.com/hyperion/index.html

Tang ZhaoHui,Jamie MacLennan (2005). Data Mining with Microsoft SQL Server, Wiley, 2005

XML for Analysis, http://www.xmlforanalysis.com/