# Integration of Data Mining and Relational Databases

Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, Usama Fayyad*
Microsoft, USA

## Abstract

In this paper, we review the past work and discuss the future of integration of data mining and relational database systems. We also discuss support for integration in Microsoft SQL Server 2000.

## 1. Introduction

Data mining techniques, based on statistics and machine learning can significantly boost the ability to analyze data. Despite the potential effectiveness of data mining to significantly enhance data analysis, this technology is destined be a niche technology unless an effort is made to integrate this technology with traditional database systems. This is because data analysis needs to be consolidated at the warehouse for data integrity and management concerns. Therefore, one of the key challenges is to enable integration of data mining technology seamlessly within the framework of traditional database systems [7].

## 2. Related Work in Data Mining Research

In the last decade, significant research progress has been made towards streamlining data mining algorithms. There has been an explosion of work (e.g., [1]) in *scaling* many major data mining techniques to work with large data sets, i.e., ensuring that the algorithms are "disk-aware" and more generally, conscious of memory hierarchy, instead of making the assumption that all data must reside in memory. Another direction of work that has been pursued is to consider if data mining algorithms may be implemented as traditional database applications. Such implementations ensure that the data mining implementations are not only disk-aware but also "SQL-aware", i.e., the implementations take advantage of the functionality provided through the SQL Engine and the API, e.g., [2]. Efforts to implement mining algorithms on top of database systems have also led to primitives such as sampling to ease the task of data mining on relational systems, e.g., see the proposal in [3].

## 3. Representing Mining Models in Databases

The progress in data mining research has made it possible to implement several data mining operations efficiently on large databases. While this is surely an important contribution, we should not lose sight of the final goal of data mining – it is to enable database application writers to construct *data mining models* (e.g., a decision tree classifier, regression model, segmentation) from their databases, to use these models for a variety of predictive and analytic tasks, and to share these models with other applications. Such integration is a precondition to make data mining succeed in the database world.

Recognizing the above fact, it is obvious that a key aspect of integration with database systems that needs to be looked into is how to treat data mining models as first class objects in databases. Unfortunately, in that respect, data mining still remains an island of analysis that is poorly integrated with database systems. Recall that a data mining model (e.g., classifier) is obtained via applying a data mining algorithm on a training data set. Although a mining model may be derived using a SQL application implementing a training algorithm, the database management system is completely unaware of the semantics of mining models since *mining models* are *not explicitly represented* in the database. But, unless such explicit representation is enabled, the database management system capabilities cannot be leveraged for sharing, reusing and managing mining models in a rich way. In particular, even if several mining models have been created, there is no way for a user or an application to search the set of available models based on its properties, indicate that a certain model should be applied to predict a column of an unknown data set and then to query the result of the prediction, e.g. to compare the results of predictions from two models.

*Author's current affiliation is with digiMine, Inc. (Usama@digiMine.com)



**Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000**

In order to effectively represent data mining models in relational databases, we need to capture *creation* of data mining models using *arbitrary* mining algorithms, *browsing* of such models (examining their structure or contents), and *application* of a selected model to an ad-hoc data set for analysis tasks such as *prediction*. Furthermore, for a column that is the result of the prediction, sufficient meta-data must be available with the predicted column so that analysis tools can interpret properties of prediction, e.g., its accuracy.

Relational database systems understand and support only relations as first class objects and so if we are to represent a data mining model in databases, it must be viewed as a "table-like" structure. However, at a first glance, a model is more like a graph, with a complex interpretation of its structure, e.g., a decision tree classifier. Thus, trying to represent a mining model as a table (or a set of rows) appears unnatural. Fortunately, this need not be the stumbling block. The key steps in the lifecycle of a mining model are to *create and populate* a model via an algorithm on a training data source, and to be able to use the mining model to *predict* values for data sets. If we can capture these steps using SQL metaphors then it would ensure that database developers are able to leverage data mining functionality without a shift in their paradigm for application development. In the next section, we outline an approach to accomplish this goal.

## 3. OLE-DB for Data Mining

OLE-DB is a well-known programming interface that allows an application to connect to and consume information from any relational data source (need not be a SQL database). This generality makes OLE-DB an ideal candidate API to extend with data mining capabilities based on the design philosophies outlined in the previous section. Microsoft has proposed OLE-DB for DM to enable data mining functionality on OLE-DB providers. The details of this specification is available online [4] and we only provide a short overview here.

The OLE DB for DM interface allows client applications to explore and manipulate the existing mining models and their properties through an interface similar to that used for exploring tables, views and other first class relational objects. All mining models are represented as table-like objects with columns, data types and extended meta-information that is needed for data mining algorithms.

Before we discuss how one can view mining models as tables, we need to recognize the data representation needs related to mining. Traditional statistical learning algorithms prefer to view a data set to consist of (attribute, value) pairs representing "cases" (observations) on a certain entity, e.g., customer. Effective data mining often requires that these cases in the training consolidate all the information relevant to the entity. Since in relational databases data is often scattered over multiple normalized tables, this creates a conceptual mismatch. In particular, cases are far better represented as nested records than flat records[1]. Traditional SQL representation also falls short is in capturing metadata on columns. To effectively derive and use a mining model, we must be able to identify properties of an attribute (e.g., discrete vs. continuous) and relationships among attributes. Accordingly, we have enabled a column in a rowset (representing a set of cases) of the four **content types**: *key, attribute, relationship, and qualifier.* The notion of key and attributes are traditional, i.e., a key uniquely identifies the case and an attribute is a property of the case. More interesting are the column types relationship and qualifier. A relationship column qualifies/classifies an attribute, e.g., a column "product type" classifies the attribute "product name". Thus, the column "product type" can be recorded as *related to* "product type". A qualifier represents a column that provides additional information on value of another attribute that can be interpreted by an OLE-DB provider. For example, a column may represent the probability associated with a predicted attribute, distribution information, domain types (discrete/continuous) etc. Such information can be invaluable in data analysis. We describe how the key steps on a mining model are supported in OLE-DB for DM:

**CREATE:** A client application can create (define) a new mining model simply by executing a *CREATE MINING MODEL* statement. This statement is very similar in style and semantics to that of a CREATE TABLE statement but contains more meta-information about the columns as indicated above. An example is presented below:

```
CREATE MINING MODEL  [Age Prediction](
[Customer ID]LONG        KEY,
[Gender]  TEXT    DISCRETE,
[Age]     DOUBLE DISCRETE PREDICT,
[Product Purchases] TABLE(
[Product Name]    TEXT    KEY,
[Quantity] DOUBLE NORMAL CONTINUOUS,
[Product Type]    TEXT    DISCRETE    RELATED
TO [Product Name]))
USING [Decision_Trees_A]
```

Note that each column has an additional meta-data specification that identifies its column type. For example, Product Type is marked as related to Product Name. The training algorithm is indicated using the USING clause. The columns that are being predicted are indicated by use of the keyword PREDICT.

**INSERT:** Once the mining model was created, it functions as an empty table, i.e., its structure could be

[1] Microsoft Data Access Components support hierarchical rowsets using the Shape Provider. See [4] for details.

browsed but queries will always return empty data. In order to be able to execute prediction using a mining model, it must be *trained* with known cases by using the *INSERT* statement that will point to the source of the training data (like source of input rows in SQL). The behavior of this INSERT statement is somewhat different than that of the traditional INSERT operation on a table. The training cases (i.e. rows) being "inserted" into the mining model are not persisted in the mining model as it is. Instead, the mining model *analyzes* the rows and builds the mining model, which could be a set of decision trees or clusters, as per the schema specified in the CREATE statement used to define the model. Once the training rows are consumed, the mining model is marked "trained" and could be used for prediction queries. Usually the physical the size of the model is quite small as it contains only the data summarization of the model rather than the original set of cases used the training.

**SELECT:** The content of the model could be browsed in various ways. In particular, the schema rowset associated with the mining model may be obtained via the following statement (see [4] for description of the mining model schema rowset):
*SELECT * FROM <my model>.CONTENT*
It is also possible to retrieve the content in XML representation consistent with the proposed PMML specification.

**PREDICTION JOIN**: Performing prediction is a simple matter of issuing SQL queries to the data-mining engine, enabled by extensions, notably the *PREDICTION JOIN* and statistics retrieval functions. Thus, the result of a prediction may be viewed as a "join" between the table representing the mining model and a data set. The following example illustrates the syntax:
*SELECT <columns to predict> FROM <data mining model> PREDICTION JOIN <new data> ON <conditions>*.
The *ON* clause ties up correspondences between a column in a data mining model with a column in the data set[2]. The result of a prediction join is always a relational result set that could be either retrieved by the client application for navigation and presentation, persisted in a table within the relational database or could be used for subsequent relational operations within the same statement.

## 4. Data Mining in SQL Server 2000

Microsoft SQL Server 2000 integrates for the first time Data Mining capabilities together with relational and OLAP database engines. The Analysis Services component of SQL Server 2000 contains a data-mining engine that is exposed through an OLE DB for DM interface. The data-mining engine is integrated in both the server component of analysis services, which includes also the OLAP engine, as well as on the client component. The data-mining engine provides two classes of scalable algorithms based on work done in Microsoft Research [5,6]: classification and segmentation (clustering). Future releases of the product will extend the repertoire of algorithms. The product provides a full GUI based console for the administration of the data-mining model including the creation, training, browsing of content and access security management. While the detailed aspects of the UI are not the central focus of this paper, the UI serves as a natural way to introduce a user to how data mining can be done naturally within the SQL Server framework. However, it should be noted that the primary contribution of our proposal is the backend components that support this UI and understand the semantics of data mining.

Creation of data mining model is done through the "Mining Model Wizard" that guides the user through four easy steps for building the basic data mining model: the selection of the table containing the cases, the selection of the algorithm (classification/clustering), the identification of the case-key column and the selection of the predictable columns and the input columns (Figure 1).

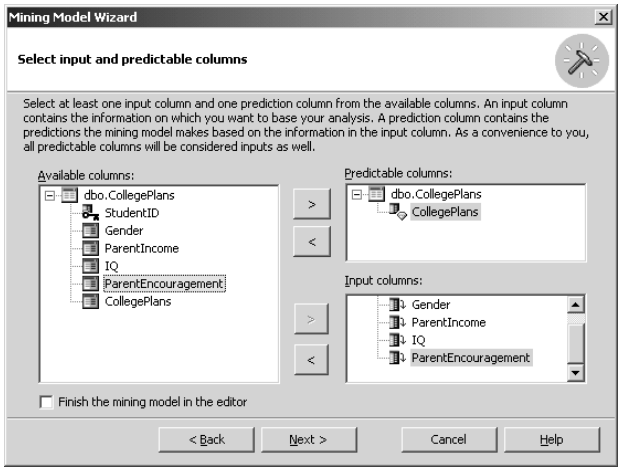


**Fig 1: Mining Model Wizard**

After the mining model has been defined and named, it is trained by simply executing the appropriate SQL statements as defined by the OLE DB for DM specification through a simple UI. Once the training is done, the user is led to a full-blown mining model editor (see Figure 2) that allows the user to enhance the basic model defined in the wizard and manipulate the properties of the model.

At this point the user can elect to browse the content of the trained model including rules and statistics by selecting the content tab where the various rules and

[2] The heterogeneous query processor of SQL Server allows queries to span both relational table and data mining models using the OPENROWSET function.

statistics could be browsed and manipulated. Figure 3 displays a view of a classifier.

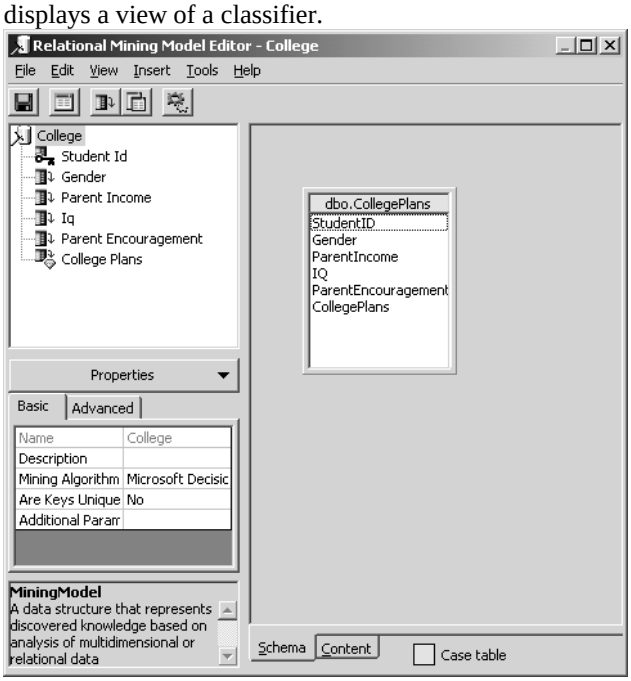


**Fig 2: Model Editor**

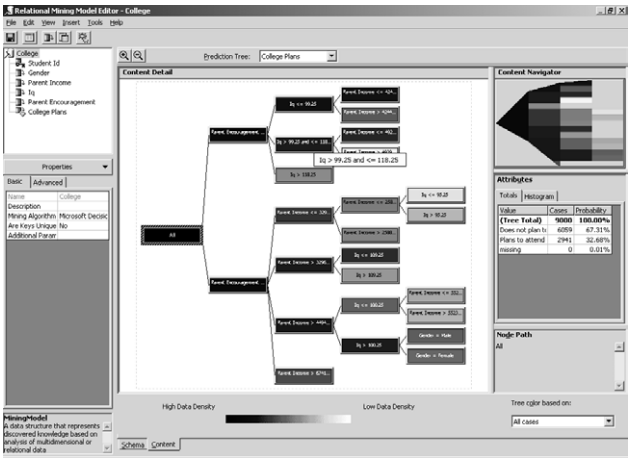

**Fig 3: Model Browsing**

The object on the upper right corner indicates the distribution of population among classes via the degree of shade in color. Since the mining model can support multiple predicted attributes, each mining model can result in multiple decision trees. A special Bayesian Dependency Network browser can be used to view the relationships among the attributes across all of the decision trees. The browser allows identification of prediction relationships and enables the user to manipulate the relationship strength slider to better identify the most important relationships.

In summary, SQL Server 2000 provides a comprehensive and open platform for the development of embedded data mining applications. The OLE DB for DM interface fits well into the relational framework and allows developers to leverage their familiarity with the SQL language and the OLE DB interfaces to facilitate application development using the data-mining technology. The easy to use administration tools ease the burden of managing the repository of data mining models and the scalable data-mining algorithms supported in the product enable the applicability of the technology to potentially a wide range of scenarios and applications.

## 5. Outstanding Challenges

OLE-DB for DM takes a significant step forward in proposing an approach that can make it easy to support the data mining process. Our proposal emphasizes the importance of being able to generate and reuse mining models effectively. While this must indeed be the first priority, it is also important to support interfaces that enable providers of data mining technologies to be able to register and integrate effectively a variety of data mining algorithms to create mining models.

Finally, there is still the issue of efficient *implementation* of data mining algorithms on the relational engine to optimize execution of integrated querying and mining [7]. Complexity of modern relational engines require us to determine judiciously the functionality that needs to be pushed down into the engine vs. the functionality that can be layered on top of the relational engine.

## 6. References


[1] Rakesh Agrawal et al.: Fast Discovery of Association Rules. Advances in Knowledge Discovery and Data Mining 1996: 307-328

[2] Sunita Sarawagi, Shiby Thomas, Rakesh Agrawal: Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Data Mining and Knowledge Discovery 4 (2/3).

[3] John Clear et al.: NonStop SQL/MX Primitives for Knowledge Discovery. *KDD 1999*: 425-429.

[4] Introduction to OLE-DB for Data Mining: http://www.microsoft.com/data/oledb/dm.

[5] Surajit Chaudhuri, Usama M. Fayyad, Jeff Bernhardt: Scalable Classification over SQL Databases. ICDE 1999: 470-479

[6] Paul Bradley, Usama Fayyad, Cory Reina: Scaling Clustering Algorithms to Large Databases. *KDD-98,* pages 9-15.

[7] Surajit Chaudhuri: Data Mining and Database Systems: Where is the Intersection? Data Engineering Bulletin 21(1) 1998.