A Framework for Data Mining in Multidimensional Databases, Part I

Alexander Russakovsky
Hyperion Solutions Corporation

Abstract. In this note we propose a framework for enabling data mining on multidimensional databases. We show that mining models and results can be treated as multidimensional entities which can thus be stored, processed and queried by the same database engine. The framework also provides isolation between the database and data mining algorithms that serve as plug-ins. It has been successfully implemented within Hyperion Essbase with multivariate regression, neural net, association rules, decision tree, clustering and naïve Bayes algorithms.

Multidimensional databases represent data as what mathematicians would call a graph of a vector function of several variables $F : D_1 \otimes D_2 \otimes \ldots \otimes D_n -> M_1 \otimes M_2 \otimes \ldots \otimes M_m$.Traditionally, the n independent variables are called "dimensions" and the m dependent ones are called "measures". Values of independent variables within dimensions are called "members", and databases are called "cubes". Dimensions may be hierarchically organized. Mathematical foundations of multidimensional database models and bibliography on the subject can be found in [R99]. Most popular commercial multidimensional databases, in historical order, are Express (now part of Oracle database), Hyperion Essbase, and Microsoft Analysis Services. Each one has its own proprietary access language although both Essbase and Analysis Services support MDX, a SQL-like query language developed by Microsoft.

Data mining activity involves using a mining algorithm to build a model given a sample dataset within the database. The model can be further applied or scored against a similarly organized set of data to produce mining results (such as prediction or classification, etc). Specification of a mining task must identify the training dataset, mining attributes and cases, and and indicate the roles of the attributes (e.g. predictor or target, etc).

In a relational database, if mining attributes correspond to columns and cases - to rows, data mining task specification can potentially use SQL combined with the roles description. In fact, some proposed languages, such as, for instance, Microsoft's proprietary DMX language, attempt to do just that. If mining attributes are not column based, the corresponding statement may be quite tricky requiring pivoting operations, auto-joins, etc.

In a multidimensional database, the dataset of interest may be arbitrarily located in the cube, and this can be expressed without complicated expressions. Rows and columns are just two different directions of navigation. In the cube, navigation is possible in many directions, for instance, along any dimension or

even diagonally. So there, when defining a dataset for mining purposes, we need to identify those directions that correspond to mining attributes and to the sequence of cases.

This naturally leads to treating task datasets, mining models and mining results as cubes, so that they are stored and server by the same database engine. To the best of our knowledge, no other vendor supports a full range of data mining operations within the same multidimensional database system.

Let us start with defining datasets, such as training data. To explain the concept, let us use an MDX expression as an example of a query on a multidimensional database. An MDX query is a mapping from the cube to a Cartesian product of certain "axes". Several dimensions of the original cube can be mapped to an axis, so that the axis consists of tuples of members of those dimensions. In MDX, the axes have fixed names: COLUMNS, ROWS, PAGES, etc. The query expression has the form

SELECT

$expression_1$ ON COLUMNS

$expression_2$ ON ROWS

...

FROM *cube*

[WHERE *slicer*]

Here $expression_i$, $i = 1, 2, \ldots$ are filters on certain dimensions, and the optional *slicer* specifies single point restrictions on some of the cube's dimensions to restrict the data set to a slice in the cube. Axes names in MDX statements are often considered as a disadvantage because it appears that the specification of the result is mixed with its presentation. We propose to consider the query result set as a new cube, with axes as dimensions, and unfix their names (i.e. allow them to be user specifiable), i.e., regard them merely as the dimension names[1]. Besides improving the general image of the language, it provides us with the ability to use particular keywords specific to data mining. We also want to be able to specify the roles of the mining attributes in the statement. For the purpose of this note, let us call the resulting language DMMDX.

Consider a sample cube tracking sales, profit, and cost for a company selling certain products and services at certain times and locations. Our goal, for example, may be to predict iPod sales as a function of sales of CDs, DVDs and iTunes subscriptions using the multivariate regression algorithm.

In the mining task specification, we represent the training dataset in the original cube as a subcube, with one of its dimensions, consistsing of mining attributes, called *Attribute* and another one, consisting of cases, called *Sequence*. There may be other dimensions in the dataset as well. These other dimensions are called "external" and are used to produce a family of models. Each member combination of external dimensions corresponds to a separate model. A typical example would be creating a separate model of iPod sales behavior for each

[1] Mapping axes to a rendering layout could be a separate clause in the language

city in the Location dimension. The entire family of models is built with one statement[2]:

CREATE MODEL SalesModel USING ALGORITHM Regression AS
SELECT
PREDICTOR AS {CD, DVD, iTunes}, TARGET AS {iPod} ON Attribute
{Time.Level (Month).Members} ON Sequence
{Location.Level (City).Members} ON Location
FROM SalesCube
WHERE {Sales}

If the above statement is successfully executed, it results in the creation of a $SalesModel$ cube which we will describe below. Recall that multivariate regression algorithm creates approximation to the model data in the form of a hyperplane $y = \sum a_i * x_i + b$.Here y is our "target", x_i are "predictors", and coefficients a_i and b are called "slopes" and "intercept" respectively. The model consists of slopes and intercept. We then form a dimension of the $SalesModel$ cube called $Model$ consisting of all predictor members and an Intercept member. Obviously, there are as many slopes as there are predictors, so we can parameterize them with predictor names. In our case the $Model$ dimension is {CD, DVD, iTunes and Intercept}. The Location axis is external in the above statement. It also becomes a dimension in the model cube. The Sequence axis from the statement is not needed for the model.

How did we end up creating the dimensions of the model cube? One possible approach is to assume that we know how the multilinear regression model looks like and can create it properly. But how to achieve it for an arbitrary algorithm? Clearly, the algorithm must direct us to do this somehow. That's where the data mining framework comes in.

With the framework, it is not really necessary for the algorithm to know about the structure of the data source and understand DMMDX or any other language. Instead, the algorithm trades description of his model structure for the ability to navigate the dataset either through the cases (sequence) or through predictors and targets (attributes) and to read and write data at current position.

The framework parses the language statements, creates the corresponding cube for storing the model, provides navigation interfaces to the algorithm and lets it drive the training process. The framework can do it iteratively for each member from the external dimension(s). However if multiple levels are present in external dimensions, it may resort to other strategies.

With such an approach, algorithms become plug-ins in the framework as long as they implement a method to communicate the structure of their models ("signature"). No other apriori knowledge of the algorithm is required from the framework to perform data mining with it. If the algorithm signature is known, the framework can also advise the users, what information is required to start a

[2] The actual syntax used in the Hyperion Essbase implementation of the data mining framework is slightly different. It is based on XML specification equivalent to the DMMDX statements presented here.

data mining procedure and even dynamically generate GUI through which the user can specify this information.

On the other hand, the algorithm does not have to know anything about implementation and access language of the database behind the framework (actually, the database does not even have to be multidimensional – it could be relational, for example, – as long as it is capable of providing the navigation interfaces). The obvious advantage to the algorithm vendor is that the algorithm can be used without any change on any data source once the database vendor implemented the data mining framework. Hyperion's implementation also allows the algorithm to execute directly within the database kernel which ensures proper performance and relieves the algorithm from having to manage memory.

Due to lack of space, we will describe here only the basic idea regarding representing of models as multidimensional entities and explain them on regression and neural net examples.

Logically, the way the algorithm would describe its model organization to the framework is as follows: it must identify, which dimensions (directions of navigation) are there in the model and how they are related to input data. For example, the multivariate regression algorithm would specify in the signature something like this:

Input : $Attribute\{Predictor, Target\}; Sequence$

Model : $Model\{Slope, Intercept, Error\}, Slope \approx Predictor.$

If an element of the $Model$ dimension is not related to one of the input mining attributes, it is supposed to correspond to a single member. In other words, the size of the $Model$ dimension in the model is determined by the framework as $size(Predictor) + 2$.

There may be cases, however, when an algorithm needs to have additional dimensions in the model, for instance, to store a matrix. Also, it may happen that the size of a model attribute may not be directly expressible via the size of any input attribute. Such model attribute is then marked "internal" to indicate that the navigational interface at model construction time should allow appending new members to the corresponding dimension as the algorithm writes model information. Consider the neural net algorithm. Assuming that it always creates one hidden layer, it must store two matrices of weights: one of size equal to $size(Predictor) \times size(Hidden)$ and another one of size equal to $size(Target) \times size(Hidden)$. The signature of such algorithm would look like this:

Input : $Attribute\{Predictor, Target\}; Sequence$

Model : $Model\{InputWeight, OutputWeight, InputBias, OutputBias\},$
$InputWeight \approx Predictor, OutputWeight \approx Target; Internal\{Hidden\}$

The algorithm signature also contains a description of algorithm's parameters for various tasks.

References

[R99] A. Russakovsky. Mathematical Foundations of Multidimensional Database Models. 1999. http://cs.russakovsky.com/multidimensionaldatamodel.pdf