By Bing Liu *and* Alexander Tuzhilin

Managing Large Collections of Data Mining Models

Data analysts and naive users alike in information-intensive organizations need automated ways to build, analyze, and maintain very large collections of data mining models.

Model building is a key objective of data mining and data analysis applications. In the past, such applications required only a few models built by a single data analyst. As more and more data has been collected and real-world problems have become more complex, it has become increasingly difficult for that data analyst to build all the required models and manage them manually. Building a system to help data analysts construct and manage large collections of models is a pressing issue.

Consider a credit-card marketing application. The credit-card-issuing company wishes to build models describing the behavior of small segments of customers, or microsegments. Examples are middle-age customers with children in college living in zip code 10012 and graduate engineering students at university XYZ. A large credit-card company might have to deal with tens of thousands of such microsegments, each involving dozens of different models. Therefore, it may need to build and support hundreds of thousands of models. Similar problems also occur in personalization applications and e-commerce.

The traditional approach is to aggregate the data into large segments, then use domain knowledge combined with "intelligent" model-building methods to produce a few good models. Intelligent means selecting the right functions and model types based on automatic algorithms and domain-expert knowledge. This approach reduces the number of models. However, it does not eliminate the need for

a large number of models in practice and is not ideal because some important characteristics of smaller models are lost in the aggregated models. The approach thus represents a compromise due to a human analyst's own limited intellectual capacity. With today's computing power, building a large number of models is not an issue. The bottleneck is the human analyst.

An initial approach to developing such tools [7] involved an application in Motorola, Inc. The system (called "Opportunity Map"), which has been in regular use since 2006, includes a set of customer-designed rule-manipulation and visualization operations. However, the approach lacks a formal foundation, and new features are constantly being added in an ad hoc manner. Model-management research should give the approach a theoretical foundation and extend it to other types of models rather than only to rules.

Our goal here is twofold. First, we want to make the argument that a generic model-management system is needed to facilitate large-scale model-building applications. And second, since model management is a relatively unexplored problem, we want to identify the main issues and problems and set up a research agenda, rather than present specific solutions, which is a separate research problem for the data mining community.

The idea of model management has been studied in the IS community since the mid-1970s [11]. However, the focus was on organizational and business issues and operations research and management-science-type models (such as mathematical programming, production and distribution, and transportation models) [5]. No industrial-strength systems were built, and little work was done on ways to manage statistical and data mining models. Interest in model management tailed off after an empirical study of 192 firms showed these firms had used about 200 models each on average in 1998 [12]. The database community also studied model management [1], but models there referred to schemas and metadata of database systems, which are quite different from data mining models.

Several data mining researchers proposed data mining query languages (such as DMQL [2] and M-SQL [10]). The main idea was to integrate data mining and database management systems and extend SQL to include operators for constrained rule mining. M-SQL also allows limited querying of mined rules. M-SQL was significantly improved in [9] by the Rule-QL language, which has a formal rule calculus based on the full set of first-order-logic expressions and is able to query multiple rulebases. The query part of the proposed work aims to expand Rule-QL to other models.

The integration of data mining and databases is called "inductive databases" [4], focusing on constrained pattern/rule mining within the database engine. However, predictive models are also important. The number of predictive models can be quite large and thus in need of effective management.

On the industrial front, Microsoft SQL Server 2005 provides model-building tools (such as decision trees, neural networks, and association rules). Model building and testing can be done through a data mining query language like SQL. However, such languages are limited, useful mainly for testing a model to determine model accuracy.

Oracle Data Mining (ODM) supports a range of data mining models and model-assessment and model-inspection capabilities. Model metadata can be queried through SQL (via the PL/SQL API). Data mining techniques are embedded in ODM's database engine; in Microsoft SQL Server, the data mining server and the database server are separate entities.

SAS acknowledged the importance of generating and managing models by adding model-management capabilities to its Enterprise Miner. Users are thus able to "register" their models with the SAS Metadata Server for subsequent retrieval. SAS also provides a Web-based model repository viewer for the user-analyst to browse the deposited models.

The Java Data Mining (JSR-73) standard [3] enables the embedding of model-management functions into applications. For example, it allows the importing and exporting of multiple model representations, as well as the querying of model metadata.

Although these initial approaches toward embedding model-management functions constitute an excellent starting point, much more needs to be done to develop a deeper understanding of model management. We propose to pursue this work further by focusing on automating a significant part of the model building, management, and analysis process for applications in which a large number of models are built.

We also note that little prior work was done on automated analysis of modelbases, yet it remains a major and challenging research topic. When dealing with a large number of models, it is necessary to analyze them using automatic techniques to find useful models, delete useless ones, identify weaknesses of models, and suggest repair actions. Manual analysis based on the experience of analysts alone is not feasible.

Building a Large Number of Models

Model management involves the following major tasks:

Building. Semiautomated or automated generation of a large number of models and the organization and storage of the models;
Analyzing. Querying and analyzing models in the modelbase; and
Maintaining. Keeping models and modelbases up to date as the environment and corresponding data change.

Traditionally, data analysts constructed models manually by choosing dependent and independent variables, along with the type and the structure of the model, then fitting the model against the data. This approach is fine for generating a limited number of models but does not scale to applications dealing with hundreds of thousands of models. Therefore, it is important to develop effective and efficient methods for generating large modelbases.

The first step in the process is to determine the data model of the modelbase and define its schema. Heterogeneous models can be organized in modelbases by grouping them based on either application or model type. In the former, models belonging to the same application are stored in the same table; in the latter, models of the same type are stored in the same table, so, for example, all decision trees are stored in a separate table, and all the logistic regressions are stored in yet another table. Although each method has certain advantages, we focus on the latter and assume that models are grouped together based on type. This approach is more natural because it is more convenient to design a schema for a single type of model than it is for multiple types; different models may require different representations and have different model properties. The schema of the modelbase may contain the following fields for supervised models:

ModelID. The key attribute uniquely identifying a model in the modelbase;
TrainData. A pointer to the data file used for building the model;
TestData. Like TrainData but pointing to the test data to compute model performance; it may be optional since a separate test data set may not always be needed, as in cross validation;
Model. The "object" in which the model is stored; and
Model property attributes. For defining properties of the model derived by accessing the model attributes; for example, in the case of decision trees, certain statistics (such as the number of nodes in a tree) can be precomputed and stored as model-property attributes, which can vary depending on model type.

This schema may need to be extended or customized to suit application-specific needs.

Once modelbase schemas are designed, the modelbase must be populated with models. A scalable approach is to generate models and populate the modelbase semiautomatically. That is, the analyst iteratively and interactively formulates requests of the form "for data set X build the models of type Y and of the form Z" where X is the data set defined by the TrainDataID identifier of the modelbase, Y is the type of model to be generated (such as decision tree and SVM), and Z is an expression specifying a template (or constraint) defining the models that should be built and stored. For example, analysts can build all the decision trees having "Purchase_Decision" as a class attribute and the "Income" variable as the root node. This approach must be complemented with various methods for selecting the sets of most important attributes, performing the right transformations to the data, and assisting analysts in imposing better and more meaningful constraints Z on the parameters of the models to be generated.

Analyzing Modelbases

Analysis aims to improve users' understanding of the models stored in modelbases. This can be done either manually (or semiautomatically) by a domain expert using analysis tools or automatically by a software system deploying model-inferencing techniques:

Manual model analysis. The manual approach includes techniques and tools that allow data analysts to examine and evaluate large collections of heterogeneous models. The most effective techniques include the modelbase query language, model-examination operators, and modelbase usage analysis and reporting tools;

The query language should support multiple modelbases of heterogeneous models (such as logistic regression, decision trees, and rules). It should also be based on first-order logic, supporting a set of model-independent and model-specific functions for extracting certain properties from these models. An example of a model-independent function is the predictive accuracy of the model; corresponding examples of queries utilizing such functions would be "retrieve all models with predictive accuracy greater than 95%" and "retrieve the model with the highest predictive accuracy." The latter query would require a self-join

and can be expressed in a modelbase calculus as:

$$\{m \mid MB(m) \wedge (\Longrightarrow n)\ (MB(n) \Rightarrow p(m) > p(n))\}$$

where *MB* is the modelbase and *p(m)* is the function returning the predictive accuracy of model *m*.

The modelbase query language should support functions pertaining to particular types of models (such as number of nodes in a decision tree or beta-coefficients in a logistic regression model). A query can then request retrieval of all decision trees with number of nodes greater than *n* or logistic regression models with at least one beta-coefficient greater than 1. The query language should also be able to access the data used for training and testing the model, leveraging this access to ask questions about the models and data.

In [8], we showed that although SQL can be used to express a large class of model-management queries, it is not sufficient for expressing all important queries and hence needs to be extended. For example, the query "find minimal association rules" (those whose lefthandside and righthandside do not contain the lefthandside and righthandside of any other rule respectively) could not be expressed with the standard SQL over the basic modelbase schema in which each association rule constitutes a model. However, it is possible to use macros over certain types of schema to express the query in SQL, though the resulting query is extremely slow.

If the underlying modelbase schema is XML-based, it is possible to use an XML-based query language (such as XQuery) to query modelbases. To do this, it is necessary to extend the language with functions pertaining to specific modelbases, as in SQL-based queries. Yet another issue is query optimization. All these issues constitute interesting research topics and are described in [8].

In addition to query languages, we expect online-analytical-processing-style browsing and examination operators to be implemented to let end users slice and dice modelbases in search of good and underperforming models. SAS Enterprise Miner already supports some of this basic functionality by providing model-browsing capabilities.

We also expect modelbase usage analysis and reporting tools to be added to manual model-analysis systems. Such tools are the equivalent of report generators in modern database management systems, helping data analysts identify the usage patterns of various models or classes of models in the modelbase.

Automated model analysis. When automatically analyzing a modelbase, the analyst looks to identify:

Underperforming models. Models whose performance needs improvement;

Dominated models. Models (dominated by other models) that can be removed from the modelbase; and

Missing new models. Models not in the modelbase but that should be added to it to enhance the modelbase's overall capability.

Identification and modification of underperforming models. Assume that the analyst measures the performance of models *m* in modelbase *M* using some performance measure $\mu(m)$ (such as the predictive accuracy of *m*). The analyst may want to identify underperforming models *m* that can be improved by being replaced with better-performing models *m′*, such that $\mu(m') > \mu(m)$. These better-performing models *m′* can be obtained from *m* through several techniques: One changes parameters of a model based on similar but more successful models. For example, assume that an SVM model *m* from modelbase *M* predicting online purchases for young customers in Chicago performs poorly. A similar SVM model $m' \in M$ for young customers in New York performs much better. The analyst can then take the kernel function and parameters from the New York model *m′*, transfer them to the Chicago model *m*, and test them on *m* to see if *m*'s performance improves. As another technique, model *m* can be replaced by a better model *m′* as follows: Assume that SVM is used as model *m* for the young customers in Chicago and Naive Bayes as model *m′* for young customers in New York and that $\mu(m') > \mu(m)$. The analyst can then try to replace SVM with the Naive Bayes model for young customers in Chicago.

Other techniques for identification and modification of underperforming models might also be available; developing them is an interesting research problem.

Identification of dominated models. Structural dominance is a type of dominance, whereby one model is better than another model in terms of structure. For example, in applications in which each association rule is treated as a single model, if we have association rules $X -> Z\ (c, s)$ (c is the confidence and s is the support) and $X, Y -> Z\ (c', s')$, and $c > c'$ and $s > s'$, then the first rule dominates the second rule, and only the first rule should be kept [6]. Developing efficient methods that would effectively remove dominated models from the modelbase represents an interesting research problem that would require studies of dominance relationships for different types of models.

Adding new models to the modelbase. In order to

identify which models are missing from a modelbase (and therefore should be added to it), we need to perform some form of automated inference on the modelbase similar to the logical inference used in logic, artificial intelligence, and deductive databases. In the most general case, we might define and compute the closure of models, as in the concept of logical implication used in logic and closure of functional dependencies in databases. Addressing model inferencing is difficult in its most general form, though it is possible to focus initially on the most tractable subproblems.

An example of such a subproblem is the development of methods producing new models m' from existing models $m' \in M$ such that m' dominates m as described earlier. This entails developing efficient methods for discovering such dominating models m', given the initial set of models M.

Maintaining Modelbases

Model performance changes over time based on the changes to the environment and to the corresponding data. In application or deployment mode, it is crucial to monitor model performance on real-world tasks. One way to do this is for the analyst to consider a model-performance measure $\mu(m)$ and a threshold value for each model m. If a model is no longer accurate, that is, if its performance is below the threshold, it should be modified or removed from the modelbase and a new model constructed based on the new data. The monitoring can be done periodically or continuously, depending on the application. In addition to monitoring the performance of individual models, the collective performance of groups of models can also be monitored by tracking each individual model's performance statistics. The traditional life-cycle issue of models is also important. Mechanisms for tracking the construction, evaluation, certification, deployment, and deletion of models should be provided to analysts and constitute interesting research topics.

Conclusion

Although several initial approaches to model management have been proposed by leading computer science and IS researchers, none covers the full scope of the model-management capabilities we have described here. We've studied this problem and identified several research issues that should be addressed. The most challenging is how to develop automated modelbase analysis tools, and researchers will need many years to explore it fully. On the other hand, many problems pertaining to model building and maintenance (such as the automated generation and storage of large modelbases and manual model analysis tools) are doable and likely to be solved much sooner.

Building and managing very large modelbases is a pressing issue. The development of model-management systems would help data analysts and naive users alike build better models, work with many more models, and make data mining models a common resource in an enterprise. This promise will make model management an even more important research topic in data mining. C

References

1. Bernstein, P. Applying model management to classical meta data problems. In *Proceedings of the Conference on Innovative Data Systems Research* (Asilomar, CA, Jan. 5–8, 2003), 209–20.
2. Han, J., Fu, Y., Wang, W., Koperski, K., and Zaiane, O. DMQL: A data mining query language for relational databases. In *Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery* (May 1996), 27–34.
3. Hornick, M., Yoon, H., and Venkayala, S. Java data mining (JSR-73): Status and overview. In *Proceedings of the Second International Workshop on Data Mining Standards, Services, and Platforms* (Seattle, Aug. 22, 2004), 23–29.
4. Imielinski, T. and Mannila, H. A database perspective on knowledge discovery. *Commun. ACM 39,* 11 (Nov. 1996), 58–64.
5. Krishnan, R. and Chari, K. Model management: Survey, future directions, and a bibliography. *Interactive Transactions of OR/MS 3,* 1 (2000); itorms.iris.okstate.edu/doc.html.
6. Liu, B., Hsu, W., and Ma, Y. Pruning and summarizing the discovered associations. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego). ACM Press, New York, 1999, 125–134.
7. Liu, B., Zhao, K., Benkler, J., and Xiao, W. Rule interestingness analysis using OLAP operations. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Philadelphia, Aug. 20–23). ACM Press, New York, 2006, 297–306.
8. Tuzhilin, A., Liu, B., and Hu, J. *Building and Querying Large Modelbases, Working Paper CeDER-05-16.* Stern School of Business, New York University, New York, 2005.
9. Tuzhilin, A. and Liu, B. Querying multiple sets of discovered rules. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Edmonton, Alberta, Canada). ACM Press, New York, 2002, 52–60.
10. Virmani, A. and Imielinski, T. M-SQL: A query language for database mining. *Journal of Data Mining and Knowledge Discovery 3,* 4 (Dec. 1999), 373–408.
11. Will, H. Model management systems. In *Information Systems and Organization Structure,* E. Grochia and N. Szyperski, Eds. Walter de Gruyter, Berlin, 1975, 468–482.
12. Wright, G., Chaturvedi, A., Mookerjee, R., and Garrod, S. Integrated modeling environments in organizations: An empirical study. *Information Systems Research 9,* 1 (Mar. 1998), 64–84.

Bing Liu (liub@cs.uic.edu) is a professor in the Department of Computer Science at the University of Illinois at Chicago.
Alexander Tuzhilin (atuzhili@stern.nyu.edu) is a professor of information systems and NEC Faculty Fellow in the Information, Operations & Management Sciences Department of the Leonard N. Stern School of Business at New York University, New York.

DOI: 10.1145/1314215.1314230