FlexiMine - A Flexible Platform for KDD Research and Application Construction*

C. Domshlak, D. Gershkovich, E. Gudes, N. Liusternik, A. Meisels, T. Rosen, S. E. Shimony

Dept. of Math. and Comp. Sci., Ben-Gurion University
P.O. Box 653, 84105 Beer-Sheva, ISRAEL
Contact e-mail: shimony@cs.bgu.ac.il

Abstract

FlexiMine is a KDD system designed as a testbed for data-mining research, as well as a generic knowledge discovery tool for varied database domains. Flexibility is achieved by an open-ended design for extensibility, enabling integration of existing data-mining algorithms, new locally developed algorithms, and support functions, such as visualization and preprocessing. Support for new databases is simple - currently via SQL queries to an INFORMIX database server.

With a view of serving remote, as well as local, users, internet availability was a design goal. By implementing the system in Java, minor modifications allow us to run the user-end of the system either as a Java applications or as a Java Applet.

Introduction

Knowledge Discovery in Databases (KDD) has become an important technology, and an extremely interdisciplinary research area (Fayyad *et al.* 1996). FlexiMine is a prototype KDD system currently being developed at Ben-Gurion University for testing techniques and algorithms for Data Mining and Knowledge Discovery and their evaluation in the context of real-life databases and users. The emphasis of this system and its software architecture is on **integration** of most KDD operations, (including database access and selection, preprocessing, data transformations such as abstraction, data-mining algorithms, and interactive visualization) and on **extensibility**. Thus, the system facilitates incorporation of new algorithms or their improved variants, and convenient extension of support to new databases or abstraction hierarchies. Yet, the system preserves a friendly and easy to use interface which enables users and domain experts to access the system from both local and remote locations, and to comment on, and evaluate, result quality.

Generally, KDD commercial systems such as MineSet (Brunk, Kelly, & Kohavi 1997) put heavy emphasis on visualization tools, query and reporting tools, in order to satisfy various user requirements. FlexiMine, on the other hand, is a research prototype, thus its emphasis on providing a testbed for evaluating new algorithms for data mining. Since such evaluation must involve real users and real databases, its architecture must be very modular and extensible. FlexiMine architecture and user-interface feature a rich selection of algorithms, with the option of activating them with various parameters for the purposes of testing. As many of the KDD algorithms are computationally intensive, it allows for their operation on parts of the database, and for observation of partial results as soon they become available, as well as lazy evaluation.

Design considerations and user-interface "flow" of FlexiMine appear in the two following sections. Although we emphasize system aspects, some original ideas were developed with this system as a testbed - presented in the algorithms section. We conclude with implementation and software engineering issues, especially those needed to provide system extensibility.

System Design and Implementation

The overall structure of FlexiMine is described in Figure 2. The system is implemented in a form of Java Application (or Applet, see below), consisting of several Java classes. The Java classes may in-turn call C functions which in-turn may contain SQL primitives and call the Informix DBMS. FlexiMine layers are:

1. **User interface layer** - responsible for interacting with the user, and for calls to various visualization utilities. Also handles security and the variance between running the system from a remote site (Java Applet) or from a local site (Java application).
2. **Intermediate layer** - (C with embedded SQL) Handles a protocol of requests, received from the user interface module. After performing the operations (query database, run data-mining algorithm or visualization), returns results to the user interface module. Design enables both database and algorithm independence. Database requests are translated into embedded SQL statements, to be executed

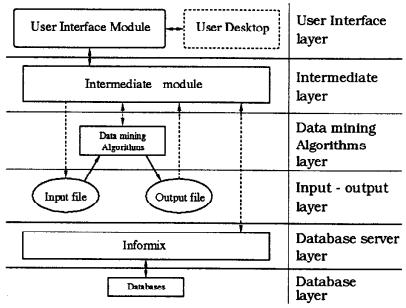

Figure 1: FlexiMine Architecture

by the database server layer. Results are rendered in a standard format file, making the data-mining algorithms independent of the database. This scheme also supports partial evaluation and pipelining.

3. **Data-mining algorithms layer** - includes decision tree learning, association rule induction, and other data-mining algorithm modules. An algorithm module consists of the implementation - a batch-mode executable file, and two Java classes, providing an interface for algorithm invocation and result interpretation and presentation.

4. **Database server layer** - Informix relational DBMS. Receives queries from intermediate module, returns results to the intermediate module.

5. **Databases** currently available are: a) student database - personal information about students, course data, and student grades. b) Hospital database - patients information: reception, tests, diagnoses, and hospitalization.

System Architecture and Flow

After logging into FlexiMine and selecting a database, the database schema is displayed, enabling the user to mark a sub-set of attributes of interest. Since most data-mining algorithms work on a single relation, one must be formed by a join operation.

The user may create new attributes which represent a form of a "denormalization" process. We call the creation of these new attributes *aliasing*. Another process at this stage is *abstraction*, which enables the user to create new abstractions to various attributes and domains. Such abstractions are extremely important in the process of data mining. A typical abstraction may be of integer grades into marks ($\{A, B, C, NC\}$).

Selection

The purpose of a selection is to produce an interesting view to the user of a given database. This view may further be aliased, and its values may (optionally) be abstracted. The result is used as input for the visualization and the data mining algorithms. A selection is assisted by a graphical interface and derived via an SQL query automatically generated by the system according to user selections.

Selections are named objects, formed by a user selection definition, as follows. A set of relations is chosen from the list of the relations of the underling database (the *relations list* on the *selection screen*). The relations contain the attributes of interest to the user, and have to share common attributes in a way that an equi-join between them is meaningful (see (Domshlak, C. et. al. 1998) for details).

Abstraction

Abstraction may be defined over domains, such as "integer", as well as on values of a specific attribute of a specific relation. The result of an abstraction is a map providing an (abstract) name or value to each of the original values. Manually defining a (named) abstraction is assisted by graphical interfaces (Figure 2). Pushbutton selection enables a choice on whether to abstract on a domain (**integer**, **float** or **fixed-length-string**) or on actual attribute values (necessitating a choice of relation and attribute). Abstraction result types are domain-based, enabling further abstractions to be defined on top of existing ones, possibly leading to abstraction ("taxonomic") hierarchies. After abstraction definition, the mechanism becomes transparent to the user, as well as to other system layers, enabling easy future expansion for automatic abstraction schemes, such as those based on automated clustering.

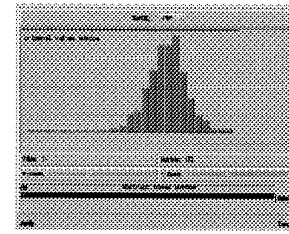

Figure 2: The Abstraction Screen

Aliasing

The aliasing mechanism enables the creation of **new** attributes, of interest to the data miner. These are treated as bona fide distinct attributes, in order to be accessible by the visualization and data-mining algorithms. Prime candidates for aliasing are "denormalized" attributes, resulting in the process of converting values in rows into values in a separate column. Consider a table containing courses grades for

students, with (stud#, course#) as a primary key. To create an alias for "final grade in the compilers course", the system should create a table with the following attributes: (stud#, final_comp, *), with * standing in for other possible attributes.

Aliasing is permissible only if the resulting table makes sense from the database theory point of view. That is, it must be the case that all new attributes are functionally dependent on the primary key of the resulting table. All attributes not depending on this new key should be eliminated - this is enforced by the system and made as user-transparent as possible (see (Domshlak, C. et. al. 1998)).

Visualization

Currently implemented are textual viewing, histograms (Figure 2), and scatter-plots. Several interactively user-selectable primary axis sorting methods are available: by value, by frequency, or scale-based (for numerical domains). The computation of the histogram is made lazy, as, due to lazy evaluation, the selected relation may not be available in explicit form at the time visualization begins. Anytime schemes, such as successive approximate partial results using sampling, are under development.

Interface with Data-Mining Algorithms

FlexiMine runs data-mining algorithms as separate processes that receives a data file, and a set of parameters. For example, an association-rule induction algorithm receives a relation, support and confidence thresholds, and optionally a set of relevant target attributes. Results are displayed to the user in text format, and saved in a temporary file, for possible use by other data-mining algorithms or visualization schemes.

Data-Mining Algorithms in FlexiMine

FlexiMine currently contains algorithms for learning ARs from a single relation and from multiple relations, probabilistic models (Bayesian Knowledge Bases), decision-trees, and meta-queries.

Distributed Induction of Association Rules

Most association-rule algorithms, e.g. (Agrawal & Srikant 1994) are a two-step process: finding all sets of items that have support above the given minimum (*frequent itemsets*), and generating the desired rules from these itemsets. In FlexiMine, we decided to implement the *Apriori* algorithm (Agrawal & Srikant 1994), as it is reasonably easy to parallelize (Agrawal & Shafer 1996), reasonably memory-efficient, and because intermediate results of *Apriori* facilitate rule filtering for "interestingness" (Silberschatz & Tuzhilin 1996), and on-line generation of the BKB model (see below).

The association rule generator is a separate task(s), accepting data in flat-file format, with output a flat rules file. Presentation is currently a list of rules, sorted by confidence, support, or both. Post-filters matching given attributes in the rule head or tail may be used. A graphical display of the rules as a (directed) hypergraph is also available in the system.

In distributing AR induction, our goal was to take advantage of commonly available computing resources, namely a set workstations on a LAN, in order to speed up generation of ARs. Easy availability and capability for using heterogeneous computing resources, was achieved by implementing the algorithms on top of PVM. We have rewritten the algorithms to take advantage of the fact that all computers share a common NFS. Each association-rule task is handled by a *master* process that creates n (dynamically determined) identical *slaves*, that collaborate on the task (Figure 3).

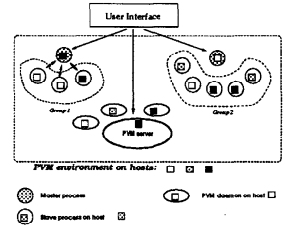

Figure 3: Process Hierarchy

Bayesian Knowledge Bases

The discovered ARs provide large amounts of potentially useful, yet sometimes unintuitive information. These may still be useful to an automated reasoning engine, especially if they possess a valid probabilistic global semantics. ARs, however, convey only local causal dependencies between sets of attribute instantiations. Graphical probabilistic models, such as **Bayes networks** (BNs) (Pearl 1988), convey local dependencies, while possessing global semantics.

BNs do not capture conditional independencies between *particular instantiations* of variables, a problem addressed by Similarity Networks (Heckerman 1991), Independence-Based Relevance (Shimony 1993), and Weighted Proof Graphs (WAODAGs) (Charniak & Shimony 1994). Other restrictions are that BNs do not allow cycles in the graph structure, and cannot represent partially known distributions. We thus use a generalization of BNs, called Bayesian Knowledge Bases (BKBs) (Shimony, Domshlak, & Santos 1997). The dependencies in the model are defined between instantiations, and cycles are allowed between variables as well as between instantiations of variables.

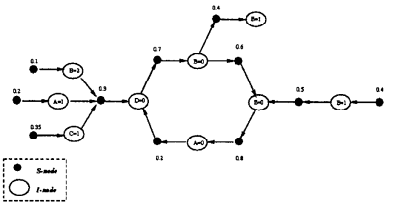

Figure 4: BKB example - variables A B, C, D, *and* E

BKBs consist of a directed graph, called a *correlation graph*, which has two distinct types of nodes (Fig. 4). *Instantiation-nodes* (I-nodes), correspond to the instantiations of the variables (several I-nodes correspond to a single node in a Bayes network - one I-node for each domain value of the Bayes network node). *Support-nodes* (S-nodes) are used to quantify the conditional dependencies between instantiations of variables (corresponding to one or more conditional probability table (CPT) entries in a BN).

Owing to Apriori decomposition, we combined the BKB on-line construction with the rule discovery part of the algorithm. We are using BKBs generated by the system to test abductive-reasoning algorithms for BKBs (Shimony, Domshlak, & Santos 1997).

Multi-relation association rules

The goal of the direct multi-relation association-rule generation algorithm is to reduce the large table created for mining ARs from multiple relations, by avoiding actual join operations whenever possible. Our scheme currently assumes that all joins are 1:n natural joins, which can be represented as a directed graph. For clarity, we consider just one path in the graph.

The algorithm consists of a pre-processing phase, and a coalescing phase (based on the Apriori-Tid (Agrawal & Srikant 1994) algorithm). During preprocessing, one follows the path and performs a "virtual JOIN", between the "1" side table and the "n" side table. This virtual JOIN does not create a resulting joined table, but does generate a **count** of how many tuples each tuple in the "1" side is connected to in the "n" side. These counts are saved and the process is repeated for each edge in the path. In the coalescing phase, the above counts are used as support counts without performing the actual JOINs, saving both space and time in comparison to the single-relation algorithm (see (Domshlak, C. et. al. 1998)).

Decision Trees

Decision trees (DT) have been treated extensively in the machine learning community. DT construction algorithms deal with noise and prune trees according to information theoretic criteria (Quinlan 1993), as well as association rule construction.

In FlexiMine we have added intermediate tree-level stopping criteria. One is based on marginal information gain - a user-selectable adaptable criterion for information gain bound. If, at some intermediate level of the DT construction, the total missing information has dropped by a fraction much larger than the "expected rate", then the tree construction procedure is stopped. The other new criterion is aimed at producing "short association rules", with a small number of attributes in the antecedent (a user-selectable tree-level limit). The algorithm will stop the DT construction at this level, *provided the information gain is above some fraction* of the a-priori missing information.

Association Rules are straightforwardly collected from the decision tree. The two standard criteria for ARs, support and confidence (Agrawal & Srikant 1994), are calculated from the weights of the paths that become rules. This method has an advantage over standard data mining algorithms because decision tree construction already prunes numerous paths that do not contain sufficient information gain.

The DT construction algorithm is implemented as a standard tool of the FlexiMine generic interface. This option of the system is used also for comparing results with the most up to date DT construction and pruning algorithms in the literature (i.e. C4.5 from (Quinlan 1993)). The initial experimental results seem quite encouraging, as the shallow trees and short ARs we produce are very good compared to C4.5.

Implementation Issues

Remote vs. Local Users. Although for privacy and security reasons, FlexiMine's user community is carefully selected, some members may need access the system from remote locations over the Internet, raising both security and user-interface problems. Currently, the user interface module is implemented as a Java application, allowing intermediate and final results to be stored in the user's home directory.

Java applications enable the above private information storage, a scheme preferable to us over storing temporary results in the database, due to their experimental (even ad-hoc) and temporary nature, and in order to keep database growth down to a reasonable size. A disadvantage of Java applications is that only users with accounts on our system can use it. Remote users may use a Java Applet version of our interface, which allows users to run the system from the Internet, but does not permit storage of any data in a user's account. Setting up local accounts for users, or distributing the Java application code to authorized remote users are

the two solutions offered in our system.

Performance Issues. In FlexiMine, when the user does not actually need all the data specified by a query, long delays are avoided by lazy computation. A typical example is when performing browsing of values in a table in order to plan a selection or an abstraction criterion. Using separate threads for retrieval and browsing allows the latter to be performed before the former is complete. Additionally, we (optionally) provide the user an estimator of fraction of retrieved results, with a capability of terminating the retrieval before completion. Estimates are currently obtained by a total-count query prior to actual retrieval, with somewhat less precise, but more efficient, schemes (sampling) under construction. System performance as a whole was significantly enhanced by storing results of queries in memory and disk (local user directory) caches.

Database Interface Generality. FlexiMine was designed (and implemented) to require no changes in the code when using a new database. The schema of the database is retrieved directly from the database server, and inserted into appropriate Java classes. SQL queries are used to extract the schema information from the catalog (Domshlak, C. et. al. 1998). Some of these queries are of particular significance, as they extract the semantics and dependencies in this database. The latter are used, in turn, in the aliasing and the selection processes, as discussed above. Catalog queries are essentially the only DBMS-dependent part of our system. Conversion to another relational database system, such as Oracle, entails little beyond a minor modification of these queries.

Summary

FlexiMine is a prototype KDD system under development at BGU for data-mining research, incorporating real-life databases, with a goal of supporting varied user types. The system emphasizes **integration** of most KDD operations, and **extensibility**. Incorporation of new algorithms or their improved variants is facilitated, as is convenient extension of support to new databases or abstraction hierarchies. Though not currently available, an interface for adding new algorithms to FlexiMine by a user *at runtime* requires only minor software changes. A user-friendly interface enables non-programmers both local and remote access to the system.

Examining several features of FlexiMine, as well as some currently incorporated data-mining algorithms in the system, we have highlighted features that are solutions to difficult system design problems, that may well be of interest to other KDD system integrators and designers. In addition, several results on data-mining algorithms (for association rules, probabilistic models, and decision trees) obtained with FlexiMine were mentioned. Several remaining system issues are of future research interest, in particular lazy evaluation and approximate computation of intermediate results (of queries and data-mining algorithms), extensibility vs. efficiency, and security issues.

Acknowledgments

This research is supported by an infrastructure grant from the Israeli Ministry of Science and Technology. We thank the anonymous reviewers for some useful ideas on future additions to FlexiMine.

References

Agrawal, R., and Shafer, J. C. 1996. Parallel mining of association rules: Design, implementation and experience. *IBM Research Report* RJ 10004.

Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In *VLDB-94*.

Brunk, C.; Kelly, J.; and Kohavi, R. 1997. MineSet: An integrated system for data mining. *KDD-97* 135–138.

Charniak, E., and Shimony, S. E. 1994. Cost-based abduction and map explanation. *Artificial Intelligence Journal* 66(2):345–374.

Domshlak, C. et. al. 1998. FlexiMine - a flexible platform for KDD research and application construction. Technical Report FC-98-04, Ben-Gurion University.

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R. 1996. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT Press.

Heckerman, D. 1991. *Probabilistic Similarity Networks.* MIT Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. 1993. *C4.5 - Programs for Machine Learning.* Morgan Kaufmann Pub., San Mateo, CA.

Shimony, S.; Domshlak, C.; and Santos, E. J. 1997. Cost-sharing in bayesian knowledge bases. In *UAI-97*, 421–428.

Shimony, S. E. 1993. The role of relevance in explanation I: Irrelevance as statistical independence. *International Journal of Approximate Reasoning* 8(4):281–324.

Silberschatz, A., and Tuzhilin, A. 1996. What makes patterns interesting in knowledge discovery systems. *IEEE Transac. on Knowledge and Data Eng.* 8(6):970–974.