Transactions on Information and Communications Technologies vol 19 © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

Constructing Data Mining Functionalities in a DBMS

Eduardo Bezerra da Silva & Geraldo Bonorino Xexéo
Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, PO Box 68511, Rio de Janeiro, RJ, Brazil, 21945-970
Email: {bezerra, xexeo}@cos.ufrj.br

Abstract

One of the main obstacles in applying data mining techniques to large, real-world databases is the lack of integration between applications and the database management system where the collection of target data is stored. In these applications, data are obtained by applying an SQL query to that database and then stored in flat files for further processing. As a result, these applications cannot benefit from previously implemented functionalities of the accessed database management system. In this paper, we present an architecture for rule extraction applications development. This architecture is constructed by adding the functionalities required in order to accomplish the extraction of rules from a database. Moreover, this architecture enables the standard and efficient construction of data mining applications for rule extraction.

1 Introduction

In our days, technology has made it considerably easy to gather and store raw data, but the analysis of such material tends to be slow and expensive. On the other hand, there is the suspicion that this not analyzed stored data can hide useful information. This leads to the urgent need of designing semi-automated methods to discover this sort of hidden information. The research area that has been developed to meet such requirement is known as *Data Mining* (DM), or *Knowledge Discovery in Databases* (KDD). The goal of this research field is to extract useful knowledge (patterns from data) from large sets of data stored in database management systems (DBMS).

Researchers and practitioners in DM have focused on the construction of efficient algorithms for the extraction of patterns from data. However, little emphasis has been given to the problem of the *interaction* among data mining applications and the DBMS. Neither has much attention been paid to the development of ways to integrate the extracted patterns and the domain knowledge (Section 3.2.2) in the DBMS itself. Indeed, current data mining techniques could be described as *file mining*, since they assume a weak coupling between the DM mechanism and the DBMS [10].

Since the beginning of this decade, several tools for data mining have been developed according to this approach. The result is that these tools access the DBMS in a non-standard. This motivates the construction of a set of functionalities common to all data mining tasks in the DBMS itself. These functionalities are embedded in an API (Application Programming Interface) for data mining. By using this API, data mining applications could communicate with the DBMS, either to extract novel patterns or to run queries on (and manipulate) previously extracted patterns.

In [10], the concept of data mining as a being equivalent to a querying process to a database and the first step towards efficient development of data mining applications are discussed. The main characteristics of a KDDMS (Knowledge and Data Management System), a system to manage data mining applications just DBMSs manage business applications, are also described.

In [8], the design and implementation of an two-level architecture for a data mining environment is presented. It consists of a mining tool and a parallel DBMS server. The mining tool organizes and controls the search process, while the DBMS provides optimal response times for the few query types being used by the tool.

In [9] Holsheimer et al. present a study of how well one can manage by using general a purpose database management systems for the discovery of association rules. A simple algorithm is presented, consisting of only union and intersection operations. Their method can incorporate inheritance hierarchies to the association rule algorithm.

In this article, we describe a data mining API to be used in the extraction of a special kind of knowledge: *rules*. The construction of this API is done in an object manager in conformity with the ODMG standard [4], the GOA++ object manager. GOA++ is extended by adding a module, that implement the additional functionalities required for the generation of rules and for their manipulation and storage. Our approach also enables domain knowledge to be defined in the GOA++ system.

In Section 2, we discuss the need of a different approach for the construction of DM applications. The data mining API implemented according to this different approach is described in Section 3. In Section 4, a toy example of an application communicating with the data mining API is presented. Finally, in Section 5, we present our conclusions and propose future developments.

2 Problems in the Current Approach

The current generation of DBMSs, particularly those that use the relational model, has been designed mainly to support information systems applications. The success of *SQL* language can be attributed to a small number of primitives that give support to the large majority of such database applications.

Current data mining applications for rule extraction also use DBMSs to access the set of target data during the extraction process. Figure 1 depicts the steps comprising data access scheme followed by such applications. In this scheme, data that will be analyzed by the rule extraction algorithm is returned from the database by a query expression that is written in the DBMSs native data manipulation language (SQL, for example). This data is stored in flat files, in the input format required by the algorithm used. The algorithm is run and the extracted rules are displayed to the user. A data mining algorithm usually presents an atypical data access behavior during its execution and may need to store partial results. If the algorithm has no access to the DBMS internal functionalities, they must be created in the application in order to run the

algorithm. Obviously, this is quite undesirable from the software engineering and performance points of view.

As illustrated in Figure 1, we can see that the database is used solely as the source of the target data set. Once obtained this set, the whole processing is done by the application (tool). Besides that, the result of this processing (i.e. the extracted patterns) are not stored in the database, in order to be available for further use by the applications, in the same way as the data.

According to [11], an analogy can be drawn between the current situation of these tools and the situation of the applications designed for information systems in the beginning of the sixties. At that time, the applications had to be constructed from the very beginning, each one having its own method of data access. Those applications could not benefit from the functionalities of the primitives, or APIs, and from the query languages provided by current DBMS.

As a result, productivity in the construction of those applications is very low. Similarly, current DM tools, in general, and rule extraction tools in particular, access data in a non-standard way. This data, in its turn, must be organized according with the requirements of the particular algorithm used.

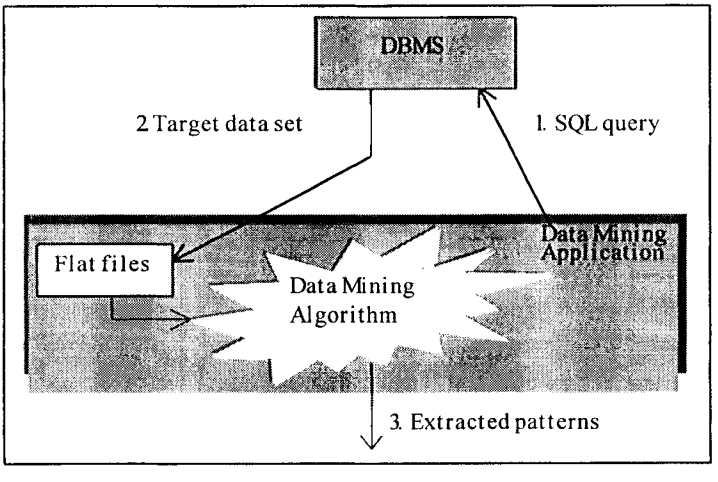

Figure 1. Current approach for database mining

Even in rule extraction applications that do have direct access to data (without using flat files) there is the assumption that data mining is only the use of machine learning algorithms to extract rules from large data sets. Moreover, the database component has the unique function of

improve the algorithm execution performance [5, 3]. However, only the performance improvement of such algorithms is not sufficient to bring forth a qualitative change in the capabilities of those applications.

Once more, an analogy with the first steps of the DBMS research is opportune: we can say that only the search for performance enhancement in database applications could not have started the research in DBMSs, in the early sixties. Instead, query languages, query optimization and transaction processing were the driving forces behind the huge development of database research. For example, the *ad hoc* nature of the query process has cleared the way for the study and development of general-purpose query optimizers. Were the queries pre-defined or were their number limited, it would suffice to construct specialized routines for their execution.

Using the functionalities provided by a DBMS, the information systems applications can be constructed in a more uniform and swift way, since all the work of data management is implemented in the DBMS, not in the application. Its is reasonable to suppose that the construction of data mining applications can benefit of the same advantages if the functionalities that are common to a certain data mining task are available in the DBMS.

3 Another Approach for Rule Extraction

Figure 2 describes our approach for the interaction between a DM application and the DBMS. In this approach, the DM application can manipulate patterns in much the same way as the information systems applications manipulate data. Thus, the DM application does not implement the data mining algorithm(s) itself. This functionality is implemented in the DBMS that communicates with this application. This DBMS provides a standard API to be used in the construction of DM applications in analogy with the API for data manipulation provided by most DBMSs. In this approach, DM applications have no concern with the extraction of patterns itself. Thus, they can dedicate themselves to other equally important aspects of the data mining process, such as giving graphical support to the data mining query construction and to the visual displaying of the results in a form that is the most adequate for the user.

The rule extraction functionalities are implemented within GOA++ system [12]. In fact is not a complete object oriented DBMS (it does not have mechanisms for transaction processing and failure

recover, for example), but an *object manager*. However, its data model is ODMG (an object-oriented database standard [4]) compliant: GOA++ has ODMG's ODL (*object definition language*) and OQL (*object query language*) languages, as well as an interface for C++ language.

In the ODMG standard, the communication between an application and the database is accomplished through the *coupling mechanism*, comprised of a library that provides database classes and functions (defined in one of the languages for which the standard is defined: C++, Java or Smalltalk). The implementation of rule extraction functionalities within GOA++ adds some functions to this library in order support the communication between a data mining application and the database. Additionally, the ODL language is extended to enable the definition of domain knowledge on the database schema.

The rule extraction functionality is implemented in one of the GOA++'s modules, named *data mining module* (DMM). The rules manipulated by DMM are statements $C_1 \rightarrow C_2$, where C_1 and C_2 are the rule's *antecedent* and *consequent*, respectively, which can be composed of conjunctions of predicates. Predicates are relationships between objects and object values in a database, in the form $X.Attr \otimes v$ or $X.Attr \otimes X.Attr$, where $X.Attr$ refers to the attribute $Attr$ of object X, v is any value in the domain of $X.Attr$, and $\otimes$ is a binary operator. A predicate can also be user-defined (see Section 3.3.2).

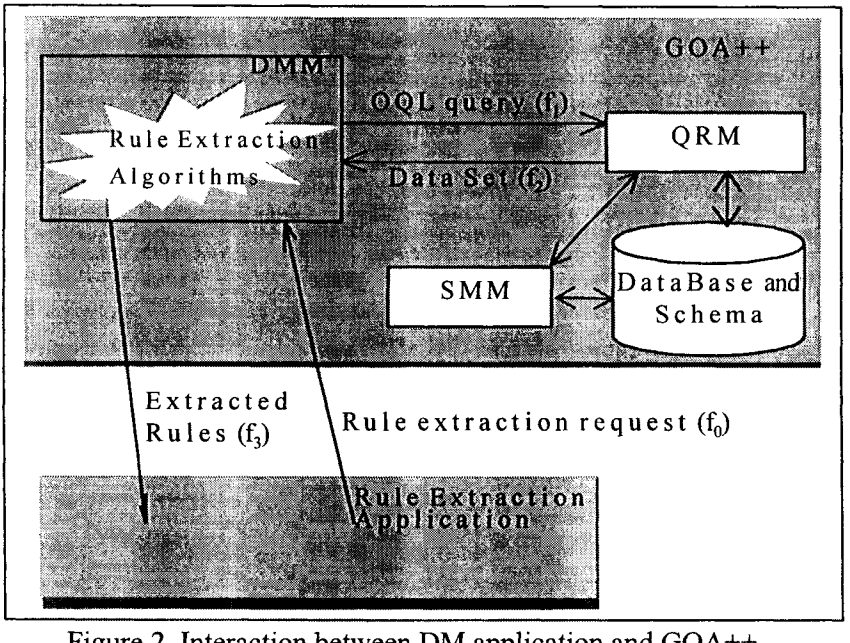

Figure 2. Interaction between DM application and GOA++

In the next sections, we describe the interactive process between a DM application and GOA++ system, This process begins with a rule extraction request being sent to GOA++ and ends up with the sending of the extracted rules to the application. In the description of this process, consider Figure 2, where there are numbered arrows (f_i). Consider, also, the following function signatures, which make part of the rule extraction API implemented by DMM (the purpose of each one is described in the next sections):

```
RuleSet& dm_query( String expression );
int dm_persist( RuleSet & );
RuleSet& dm_retrieve( String RuleSetName );
int dm_delete( String RuleSetName );
int dm_delete( RuleSet& );
```

3.1 Sending a rule extraction request

The process begins when an application sends a rule extraction request to the GOA++ system. This request is made through the *dm_query* function. This function receives a *String* object as a parameter. This object defines a *query expression* for rule extraction, whose general syntax is the following:

```
RuleQuery ::=  extract RuleType rules RuleSetName in
               TargeDataQuery
               [using DomainKnowledgeName]
               [template RuleTemplate]
               [with RuleParametersSpec]
```

This expression is part of the GOA++'s data mining query language [12]. Once the query expression is received through *dm_query* function (arrow f_0), DMM parses this expression and gets the values of each non-terminal of the expression.

3.2 Retrieving the target data set

The target data set to be used in the rule extraction is specified, in the rule extraction request, by the non-terminal *TargeDataQuery*.

This non-terminal can be specified in the expression by an OQL query. This OQL query can be (1) a *select...from...where* query or (2) the name of a database collection (previously defined in the database schema).

In both cases, the task of retrieving corresponding objects from the database is done by the GOA++'s *query resolution module* (QRM) [12]. All the DMM has to do is to convey the OQL query to this module (arrow f_1). Once determined by the QRM, these objects are sent to the DMM (arrow f_2). QRM uses the services of another GOA++ module, the *schema manager module* (SMM), which manages domain knowledge and schema information.

3.3 Extracting rules

3.3.1 Choosing the rule extraction algorithm

Once the DMM has received the target data subset, it has to know what type of rule is to be extracted from the data, in order to choose correct rule extraction algorithm. This information is provided by the value of *RuleType*. The rule types that can be currently extracted by DMM are discriminant rules, characteristic rules, classification rules and association rules. See [7] for a more detailed description about these rule types.

3.3.2 Input information for the rule extraction algorithm

The algorithms for rule extraction implemented in the DMM require some input information to accomplish their task:

1. algorithm's specific parameters (RuleParametersSpec)
2. shape of the extracted rules (RuleTemplate)
3. domain knowledge (using DomainKnowledgeName)

Once this information has been determined by parsing the query expression (sent to the DMM through the function *dm_query*), the DMM can run the particular rule extraction algorithm. The rule extraction algorithms implemented in the DMM are well known and have been studied extensively in the literature [1, 2, 6].

The remaining of this section describes each one of these input information, and the how they are obtained and manipulated by the DMM.

The algorithm's specific parameters are specified in the non-terminal *RuleParameterSpec*, which describes pairs (*parameter, value*) that shall be used by the rule extraction algorithm to bind the search space. These parameters vary according to the rule type specified in *RuleType*.

The non-terminal *RuleTemplate* specifies the general form of the rules to be extracted. By using this, one can both bind the rule search space and improve the interestingness of extracted rules.

The schema of a GOA++ database contains information about the form of this database's content, such that the specification of class names, its properties, attribute types, etc. This kind of information is used by GOA++'s QRM to resolve OQL queries.

Analogously, the rule extraction algorithm can use *domain knowledge*. Domain Knowledge can be described as some type of additional information on the application domain, which is provided by the domain specialist. Its main functions are (1) bind the pattern search space and (2) express the user preference by some type of generalization whose presence would be interesting in the patterns being generated.

In our approach, one can define domain knowledge in GOA++ database schema, along with class definitions. Thus, the DMMs algorithms can access this domain knowledge in the same way QRM access class definitions in order to solve an ordinary OQL query.

In order to define domain knowledge in the database schema, we use a *domain knowledge module* (DKM). By using a DKM, the data mining application developer can, for example, (1) define a predicate (concept), or (2) define generalization and/or specialization concepts from other defined concepts. The DKM general syntax is presented bellow:

```
DomainKnowledgeModule ::= DomainKnowledgeName '{'
                              DomainKnowledgeItem , ...
                          '}'
```

The ellipses in syntax point out to the fact that a DKM may contain several *domain knowledge items* (DKI). A DKI defines a concept that can be used by the rule extraction algorithm in the generation of rules.

By using the clause [***using*** *DomainKnowledgeName*] of the rule extraction request, we can specify the DKM to be used in the rule extraction task.

Consider, as an example, the definition of DKM named *DK_Customer*, shown bellow:

```
DK_Customer{
        good_customer: x in Customer,
          sum( coll( x.boughtProducts, price ) ) > 100,000;
}
```

For the sake of simplicity, only one DKI has been defined in this DKM. This item defines the predicate *good_customer* over the class *Customer* (assumedly already defined in the same database schema where this DKM is being defined). This predicate is a concept that defines good customers as those for whom the sum of products purchased surpasses 100,000. Once defined, this DKM can be referenced in a rule extraction request and its DKIs can be used in the extracted rules definition. An example of rule that uses the item *good_customer* is ***X.good_customer → X.age > 30***

3.3.3 Representing rules as database objects

Once extracted from the target data set, the rules must be sent to the data mining application in a format that can be easily manipulated. To solve this, we define a class for the representation of a rule set: *RuleSet.* The definition, in ODL language [4], of this class is given bellow:

```
class RuleSet
{
 attribute String name;                    // rule set name.
 attribute Set< GenericRule > ruleSet;     // rule set.
 attribute RuleType type;                  // type of rules.
};
```

We also define a class to represent a generic rule, named *GenericRule*, from which all the classes for specific types of rule derive:

```
class GenericRule
{
 attribute List< Predicate > antecedent;
 attribute List< Predicate > consequent;
};
```

The pre-defined classes for the representation of specific rules (one for each rule type enumerated in Section 3.3.1) are *AssociationRule*, *CharacteristicRule*, *ClassificationRule*, and *DiscriminantRule*. For example, the definition of the class *AssociationRule* is the following:

```
class AssociationRule: GenericRule
{
 attribute double confidence;
 attribute double support;
};
```

All these class definitions presented above (RuleSet, GenericRule, AssociationRule, etc) have global scope, which means that they are visible by any schema defined in GOA++.

In order to send the result of the rule extraction to the application, an object of the class *RuleSet* is instanciated by the DMM. Its attribute named *ruleSet* is filled with objects pertaining to one of the specific classes designed for rule representation.

3.4 Sending mining results to the application

The RuleSet object is sent to the data mining application (arrow f_3) in order to enable the application to access each one of the extracted rules.

3.5 Storing mining results

By the time the application finishes processing the rules, the set of rules can be stored permanently in the database. This is accomplished through a call to the function *dm_persist* by the application. This function has a parameter of type RuleSet (Section 3.3.2), which identifies the rule set to be stored. In this case, the rule set remains in the database when the application finishes. This rule set can be further retrieved (by calling the *dm_retrieve* function) or removed (*dm_delete* function) by the application. If the application doesn't call function *dm_persist*, the rule set is removed from GOA++ when the application disconnects from the database.

4 A Toy Example

Let us give an example of the communication process between a DM application and the GOA++'s *data mining module*. In this example, the data mining task done by the application is that of extracting *association rules* [2]. An association rule describes dependency relationships among objects.

In this example, assume the user of the data mining application has constructed a query to assess the relation between a company's male good customer (see definition of the predicate *good_customer* in Section 3.3.2) and his age.

The way in which the user constructs this query depends on the application. It can be constructed, for example, using a QBE (Query By Example) interface, where the query is defined graphically. Once the user issues the rule extraction command, the application maps the graphical representation of the rule query to the rule extraction request described in Section 3.1. The application, then, sends the rule extraction request to the GOA++'s data mining module through the function *dm_query*. The following shows a possible call to this function:

```
dm_query("extract association rules AssocCustomer as
                select x from x in Customers where x.sex = 'M'
                template  consequent contains x.age,
                                antecedent contains x.good_customer
                using DK_Customer
                with support >= .01, confidence >= 0.5");
```

In this call, the target data set is specified through an OQL query that selects all male customers (*x.sex = 'M'*) in the *Customers* database collection. From this target data set, the DMM extracts association rules relating the attribute *age* of class *Customer* and the predicate *good_customer*, defined in *DK_Customer*. Moreover, the request specifies minimum values of support and confidence to be used by the rule extraction algorithm.

Once the rule extraction request is sent to GOA++, the process described in Section 3 takes place and eventually leads to the extraction of rules according to the rule extraction request. Finally, the results (database objects representing the extracted rules) are sent back to the application.

5 Conclusions

In this article, we have analyzed the disadvantages of the current approach used by most data mining applications about the interaction with the DBMS. A different approach is described, by implementing syntactic and functional extensions (data mining API) to the GOA++ object manager, in order to give a more effective support to such applications. Parts of the data mining API have already been implemented over the GOA++ object manager, proving the feasibility of our approach. The idea of constructing a data mining API in a DBMS is not new and the literature lists some works on this subject (see Section 1). However, the approach here presented can incorporate generated rules in the database itself. Besides, domain knowledge can also be defined in the database schema. None of the previous works present this possibility. This approach provides an efficient and effective mechanism for learning various kinds of rules from a GOA++ database.

With the assistance of domain knowledge about concepts, data relevance, and expected rule forms, our approach integrates database operations with the learning process and provides a simple, efficient way of learning quantitative rules from large databases. Moreover, as the rule extraction mechanism is incorporated in the DBMS, there is a higher degree of standardization and productivity in the development of data mining applications.

However, some questions require further study. Data mining applications face challenging problems from real-world databases, which tend to be dynamic, incomplete, redundant, noisy, sparse, and very large. These problems need some techniques for handling them.

We are currently studying one of these questions: keeping the consistence between the target data and the rules extracted from them. Some techniques (used in active and temporal databases) are being studied to provide the automatic updating and versioning of the rule sets generated and stored in the database, taking into account the alterations in the data that originated them.

Additional study is also needed to provide adequate support to different types of patterns generated by data mining tasks, such as grouping, deviation, etc. It is still an open question whether it is possible or not to achieve this integration in a single data mining API.

References

[1] R. Agrawal, M. Mehta, R. Srikant, John Shafer, Andreas Arning, Toni Bollinger. "The Quest Data Mining System". In Simoudis et al. [13], PP. 244.

[2] Rakesh Agrawal, R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". J. Bocca and M. Jarke and C. Zaniolo editors, In: 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago, Chile, pp. 487-499, 1994, Morgan Kaufmann Publishers.

[3] R. Agrawal and Kyuseok Shim. "Developing Tightly-Coupled Data Mining Applications on a Relational Database System". In Simoudis et al. [13], pp. 287.

[4] R. G. G. Cattell, Object data management: object-oriented and extended relational database systems, Addison-Wesley Publishing Company, Inc., 1994.

[5] S. H. Freitas and A. A. Lavington. "A data-parallel primitive for high-performance knowledge discovery in large databases", Internal Report CSM-242, University of Essex, UK, may 1995.

[6] J. W. Han and Y. D. Cai and N. Cercone. "Data-driven discovery of quantitative rules in relational databases", IEEE Trans. On Knowledge And Data Engineering, 5:29-40, 1993.

[7] J. W. Han, Y. Fu and O. Zaiane. "DMQL: A Data Mining Query Language for Relational Databases". In: *Proceedings of the SIGMOD'96 Workshop On Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, pp. 547-559, Montreal, Canada, 1996.

[8] M. Holsheimer and M. L. Kersten. "Architectural Support for Data Mining", Technical Report, CWI, PO Box 94079, 1090 GB, Amsterdam, Netherlands, 1994.

[9] Marcel Holsheimer, Martin L. Kersten, Heikki Mannila and Hannu Toivonen. "A perspective on databases and data mining" pp. 10, Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, April, 1995.

[10] T. Imielinski and H. Mannila. "A database perspective on knowledge discovery". *Communications of the ACM*, 39:58-64, 1996.

[11] Tomasz Imielinski and Aashu Virmani and Amin Abdulghani. "DataMine: Application Programming Interface and Query Language for Database Mining". In Simoudis et al. [13], pp. 256-261.

[12] Renato C. Mauro and Eduardo Bezerra et al.. "GOA++: Technology, Implementation and Extentions to the Object Management Services". In: Proceedings XIII Brazilian Symposium on Databases}, pp. 272-286, Fortaleza, Ceará, october, 1997.