An analysis of the integration between data mining applications and database systems

E. Bezerra, M. Mattoso & G. Xexéo

COPPE/UFRJ – Computer Science Department, Federal University of Rio de Janeiro, Brazil.

Abstract

In this paper we present a classification of integration frameworks found in the literature for the coupling of a data mining application and a DBMS. We classify the database coupling in several categories. Along these categories, we analyse several issues in the integration process such as degree of coupling, flexibility, portability, communication overhead, and use of parallelism. We also present the trade-off of using each one of the integration frameworks and describe the situations where one framework is better than the other. We also describe how to implement DBMS integration in several Data Mining methods and we discuss implementation aspects including parallelism issues for each one. We analyse these solutions and show their advantages and disadvantages.

1 Introduction

The initial research on data mining concentrated on defining new mining operations and developing algorithms to solve the "data explosion problem": computerised tools to collect data and mature database technology lead to tremendous amounts of data stored in databases. Most early mining systems were developed largely on file systems and specialised data structures and buffer management strategies were devised for each algorithm. Coupling with database systems was, at best, loose, and access to data in a DBMS was provided through an ODBC or SQL cursor interface [1],[2],[4]. Recently, researchers have started to focus on issues related to integrating mining with databases.

This paper tries to classify the various architectural alternatives (here named integration frameworks) for coupling data mining with database systems. We try to classify the current frameworks of integration into three different ones:
conventional, tightly coupled and black box. The conventional category comprises loosely coupled integration where most of the data mining activities reside outside the DBMS control. In the tightly coupled category, additionally to SQL statements, part of the DM algorithm can be coded through stored procedures or user defined functions. This category also offers special services from the DBMS such as SQL extensions (data mining primitives). Finally, in the framework category, named black box, the DM algorithm is completely encapsulated into the DBMS.

Along with the frameworks proposed in this work, several issues related to the integration process are discussed, such as degree of coupling, communication overhead, flexibility, portability, and use of parallelism. We also present the trade-off of using each one of the frameworks of integration and describe the situations where one framework is better than the other. To better clarify our classification, we also describe how to implement DBMS integration in several Data Mining methods, namely, Rule Induction, Bayesian Learning and Instance Based Learning. For each one of these methods, we discuss implementation aspects including parallelism issues. We analyse these solutions and show the advantages and disadvantages of each one.

This work is organised as follows. Section 2 presents our frameworks, describing the characteristics of each proposed category. In Section 3, we discuss some issues about these frameworks, namely communication overhead, portability and use of parallelism. Section 4 focuses on tightly coupled frameworks and describes implementation issues on several data mining methods for classification regarding this framework. Finally, Section 5 presents our conclusions.

2 Integration frameworks

In this Section, we propose a classification for the development environment of Data Mining (DM) applications into three frameworks, namely, (i) conventional, (ii) tightly coupled and (iii) black box. Although we describe these frameworks separately, it should be noted that they are not exclusive, meaning that a mining application developer can use one or more of these frameworks in developing an application. The next three Sections describe the main characteristics found in these frameworks.

2.1 Conventional framework

In this framework, also called loosely coupled, there is no integration between the DBMS and the DM application. Data is read tuple by tuple from the DBMS to the mining application using the cursor interface of the DBMS. A potential problem with this approach is the high context switching cost between the DBMS and the mining process, since they are process running in different address spaces [5]. Although there are data block-read optimisation in many database systems (e.g.
Oracle [11], DB2 [9]) where a block of data is read at a time, the performance of the application could suffer.

The main advantage of this approach is greater programming flexibility, since the entire mining algorithm is implemented in the application side. Also, any previous application running on data stored in a file system can be easily changed to work on data stored in the DBMS in this framework. One disadvantage of this framework is that no DBMS facility is used (parallelism, distribution, optimised data access algorithms, etc.). Additionally, the main disadvantage of such a framework is the data transfer volume. For some DM applications, particularly in the classification task, it is simply unnecessary to move all the mining collection to the DM application address space. In other words, the DM algorithm can, in some cases, run using only a statistical summary of the entire mining collection.

Yet in this framework, we may find applications that use cache functionalities. This approach is actually a variation of the above one, where after reading the entire data, a tuple at a time, from the DBMS, the mining algorithm temporarily caches the relevant data in a look-aside buffer on a local disk. The cached data could be transformed to a format that enables efficient future accesses by the DM application. The cached data is discarded when the execution completes. This method can improve the performance. However, it requires additional disk space for caching and additional programming effort to build and manage the cache data structures.

2.2 Tightly coupled framework

In the tightly coupled framework, data intensive and time-consuming operations are inserted in the DBMS. This framework can be subdivided into two approaches, namely, the SQL aware approach and the extensions aware approach. Let us discuss each one of them in the next two Sections.

2.2.1 SQL aware approach

This approach comprises DM applications that are tight-coupled with a database SQL server. The time consuming and data intensive operations of the mining algorithm are mapped to (standard) SQL and executed into the DBMS. By using this approach, the DM application can gather just the statistical summaries that it really need instead of transferring the whole mining collection to the application address space.

There are several potential advantages of an SQL aware implementation. For example, the DM Application can benefit from the DBMS indexing and query processing capabilities, thereby leveraging on more than a decade of effort spent in making these systems robust, portable, scalable, and concurrent. The DM application can also make transparent use of the parallelization capabilities of the DBMS when executing SQL expressions, particularly in a multi-processor environment.
2.2.2 Extensions aware approach

In this approach, lie the applications that use database extensions, like stored procedures and some data types found in some object-relational and object-oriented database systems. In this approach are also the applications that use SQL extensions that suit DM core operations. Typically, the goal of using one or more of these extensions is to improve the performance of the DM algorithm. The remaining of this Section describes some works of the literature that fall in this approach.

2.2.2.1 Internal functions By internal function, we mean a piece of code written as a user-defined function (e.g. IBM DB2/CS [9]), or as a stored procedure (e.g., Oracle PL/SQL [11]). In this approach, part of the mining algorithm is expressed as a collection of user-defined functions and/or stored procedures that are appropriately used in SQL expressions to scan the mining relation. So, part of the algorithm processing occurs in the internal function.

The internal functions execute in the same address space as the database system. Such an approach was used in [5] (user-defined functions) and in [17] (stored procedures). The processing of both user-defined functions and stored procedures is very similar. In [4], a series of comparisons are made in relation to implementing association rules using both user-defined functions and stored procedures, and it is found that user defined functions are a factor of 0.3 to 0.5 faster than stored procedures but are significantly harder to code.

Internal functions aim at transferring data-intensive operations of the DM algorithm to the database system. Therefore this approach can benefit from DBMS data parallelism. The main disadvantage in this approach is the development cost, since part of the mining algorithm has to be written as internal functions and part in the programming host language [5].

2.2.2.2 Data types In this approach, DM applications use some data types provided by the DBMS to improve the algorithm execution time. A good example of this approach is the bit-precision integer data type provided by the NonStop SQL/MX system [19]. This type allows such columns of the mining collection to be represented in the minimum number of bits. In other words, the use of this data type allows an attribute whose range of values is 0 to $2^{b-1}$ to be stored in $b$ bits. This reduces significantly I/O costs. In [19], we find that around 80 percent of the attributes in a typical mining problem have cardinality of 16 or less. In such situations, using bit precision integers can result in significant savings in both storage and I/O costs.

Another example [4] of using a data type to improve the mining process is the work that describes the use of a BLOB (Binary Long Object) to implement an association rule discovery algorithm more efficiently.

2.2.2.3 SQL extensions Here, the application takes advantage of extensions to the database system SQL language to accomplish its mining task. The SQL is extended to allow execution of data intensive operations into the DBMS, in the form of primitives. This approach aims at leveraging the communication
overhead between the application and the DBMS. The works [7],[8] propose the
Alpha Primitive. This primitive can be used to implement algorithms in the
classification task. In [12],[15] other primitives, equivalent to the Alpha
Primitive, are proposed. In [19], in addition to data types extension, the NonStop
SQL/MX system also presents primitives for pivoting (transposition), sampling,
table partitioning and sequence functions.

One advantage of using SQL extensions over the (standard) SQL aware
approach (Section 2.2.1) is that same information (statistical summaries) that
would be obtained through a sequence of SQL commands can be obtained by
sending a shorter number of requests to the DBMS. Another advantage, maybe
more important, is that, as the data intensive operations are encapsulated in a
primitive and built deeply into the database engine, its execution can make use of
all DBMS capabilities, hence improving efficiency.

2.3 Black box framework

In this approach, DM algorithms are fully embedded into the DBMS. The
application sends a single request to the DBMS asking for the extraction of some
knowledge. Often, this single request is actually a query expression. The
expression specifies the kind and source of the knowledge to be extracted through
mining operators. For instance, the query language DMQL [3] extends SQL with
a collection of operators for mining several types of rules. Another proposal of
extension is the M-SQL language [16], which has a special unified operator
called MINE to generate and query a whole set of propositional rules. The MINE
RULE [18] is used in association rule discovery problems. In [6], we find a data
mining API to be used in the extraction of rules. The construction of this API is
done on top of an object manager that is extended by adding a module that
provides the functionalities required for the generation of rules and for their
manipulation and storage.

The main disadvantage of this framework is due to the fact that “no data
mining algorithm is the most adequate for all data sets”. Therefore, there is no
way for the mining algorithm implemented in the DBMS to be the best in all
situations.

3 Some issues on the framework

From the previous Section, it can be seen that, out of the three frameworks
presented, the tightly coupled approach can combine the best use of DBMS
services. In the conventional framework, the DM application does not benefit
from DBMS services at all. In the black box framework, the DM application may
have the best execution time, however it is restricted to use the algorithm
implemented in the server (DBMS).

In this Section, we discuss the frameworks in more detail, analysing several of
its aspects. We also discuss the other frameworks, but we give focus primarily on
the tightly coupled framework, since this is the one in which the interaction aspects between the DM application and the DBMS are more complex. In Section 3.1, we analyse the frameworks under portability issues. The communication overhead between the DBMS and the mining application is considered in Section 3.2. Finally, in Section 3.3, we discuss some aspects of using DBMS parallelism.

3.1 Portability

In the SQL aware approach, the data manipulation language (DML) of the DBMS (e.g. SQL-92) is used. Thus, this is the most portable framework. In the extensions aware approach, the DM application uses non-standard DBMS features (DML extensions, object-relational extensions, stored procedures, internal functions, etc), making more difficult to develop a portable algorithm.

The black box framework presents no portability at all. In spite of the application having to send only a single command to the DBMS requesting some kind of knowledge to be mined, the internal DM algorithm usually relies on specific DBMS architectural features. Thus, the application becomes very dependent on the particular database system [13].

3.2 Communication overhead

In the conventional framework, the communication overhead is maximised, since the mining relation is accessed a tuple at a time. However, when a caching mechanism is used the communication overhead can be reduced significantly. In [4], an approach called Cache-Mine is used, and the improvement of performance due to the reduction of communication overhead is shown to be very good.

When considering the tightly coupled framework, within the SQL aware approach, the communication overhead between application and DBMS can be greater when compared to the extensions aware (primitives) approach. Communication overhead is reduced in the extensions aware approach because a lesser number of requests to the DBMS is need to accomplish the same operation. As the primitives are built deep into the DBMS, more time-consuming operations are executed at the DBMS, and the communication overhead is diminished compared to the SQL aware approach.

In the black box framework, communication overhead is minimum, since the entire DM algorithm is inside the DBMS (only the discovered knowledge is returned to the application after a single request has been delivered).

3.3 Performance improvement with DBMS parallelism

Some well-known techniques may be used to improve DM algorithm execution time, such as: sampling (reducing the number of processed tuples), attribute selection (reducing the number of attributes), and discretization (reducing the number of values, which, in turn, reduces the search space). These alternatives reduce the quantity of data being mined, therefore improving execution time. However, discovered knowledge can be different than the one discovered using
Data Mining II

the original data set. Additionally, in most cases, sampling reduces the accuracy of the discovered knowledge and both attribute selection and discretization can either reduce or improve the accuracy of the discovered knowledge, depending on the data set to be mined.

A parallel DM algorithm can, in principle, discover the same knowledge than its sequential version. Therefore, when an application needs to improve a DM algorithm without loosing in the accuracy of the discovered knowledge, a natural solution is to use parallelism.

In a parallel DBMS, let \( N \) be the number of tuples of the mining relation and \( p \) be the number of available processors. Any sequential mining algorithm executes on \( \Omega(N) \) time to process data. Parallelism can reduce this lower bound to \( \Omega(N/p) \). Let us analyse the use of DBMS parallelism in the tightly coupled and black box frameworks.

Regarding the tightly coupled framework, within the SQL aware approach, parallelism can be used automatically, because the DM application relies on the DBMS parallel queries execution. In the extensions aware approach, in general, internal functions (Section 2.2.2.3) can execute in parallel. The DM primitives are also constructed having parallelism in mind [8],[12],[15]. Indeed, since the primitives are constructed as extensions to the query language of the database system, the database system optimiser can be modified to efficiently execute the operations of the primitives (see [8] for more details).

Considering the black box framework, it can also make high use of the DBMS parallelism, since the DM algorithm is implemented into the DBMS. But here, optimisation of the mining operators is much more difficult. Indeed, most papers that propose such operators do not consider this problem at all.

4 Implementing a DM algorithm using tightly coupled framework approaches

To clarify our discussion about the tightly framework, the one in which the integration issues are more evident and complex, in this Section we describe how to implement some data mining methods in such a framework. Our focus is on methods for the Classification Task [8], which is by far the most studied task in Data Mining. In this task, the goal is to generate a model in which the values of some attributes (called predicting attributes) are used to predict the value of the goal attribute (class). This model can be used to predict the class of tuples not used in the mining process. In the next Sections, we describe what tightly coupled approaches can be used to implement well known classification methods: Rule Induction, Bayesian Learning and Instance Based Learning.

4.1 Rule Induction

In the Rule Induction (RI) method, a decision tree is constructed by recursively partitioning the tuples in the mining collection. In its most basic form, the method selects a candidate attribute whose values will be used in the partitioning. This attribute is then used to form a node of the decision tree, and the process is
applied again as if this node were the root. A node becomes a leaf when all the
tuples associated with it are of the same class (have the same value for the target
attribute). When the tree is grown, each path from its root to a leaf is mapped to a
rule. Here, we describe how the most time-consuming phase of this method, the
tree expansion, can be implemented using the SQL aware (Section 2.2.1)
approach and the extensions aware approach (Section 2.2.2).

In the SQL aware approach, the following SQL selection expression is shown
to provide all the statistical resumes need to implement most RI algorithms [12]
[14]: SELECT a_i, a_c, COUNT (*) FROM R WHERE C GROUP BY a_i, a_c.
This query is from now on called Count By Group (CBG). In such an expression,
R is the mining collection, C is some condition involving the candidate attributes,
a_i is one of the candidate attributes, and a_c is the target attribute. Through several
calls to this SQL expression, the DM application can get what it needs to generate
the decision tree.

By using the extensions aware approach, more specifically, some SQL
extension to get the statistical resumes needed for RI method, communication
overhead can be improved. This characteristic is explored in [8],[15] where the
Alpha Primitive and the PIVOT operator are proposed, respectively. Both of these
are constructions whose goal is to get the statistical resumes for all the candidate
attributes in just one call to the DBMS.

4.2 Bayesian Learning

This method first finds out the conditional probabilities $P(a_i | a_c)$ for each
predicting attribute $a_i$ and the prior probability $P(a_c)$ for the target attribute $a_c$. By
using these probabilities, one can estimate the most probable value of the target
attribute in the tuple to be classified [20]. Here, the most time-consuming phase is
the one where these probabilities are calculated. Let us describe how to determine
such values by using the tightly coupled framework.

Let $N$ be the number of tuples in the mining collection, and $N(\text{predicates})$ be
the number of tuples in the mining collection that satisfies the list of conditions in
predicates. The prior probability $P(a_c = c)$ for each value $c$ of the target attribute
$a_c$ can be obtained by $P(a_c = c) = N(a_c = c)/N$. For each discrete attribute, we can
calculate its conditional probability using $P(a_i = x | C = c) = N(a_i = x, a_c = c)/N$. If
the attribute is numeric, we use: $P(a_i = x | C = c) = \frac{1}{\sqrt{2\pi \sigma^2}} e^{-\frac{(x - \mu)^2}{2\sigma^2}}$, where $\mu$ and $\sigma^2$
are the standard deviation and the variance over the values of $a_i$, respectively.

The question arises on how to determine $P(a_c = c)$ and $P(a_i = x | a_c = c)$ using
an SQL expression. For $P(a_c = c)$ and $P(a_i = x | a_c = c)$, $a_i$ discrete, one can use the
SQL aware approach (CBG, presented in Section 4.1), or the extension aware
approach (Alpha [7] primitive, PIVOT operator [15]). For $P(a_i = x | a_c = c)$, $a_i$
numeric, one can use the SQL-92 AVG and STDDEV aggregate functions (see [7]
for more details).
4.3 Instance Based Learning

This method considers that tuples in the mining collection are points in the $R^n$. Each attribute is treated as a dimension of this n-dimensional space. A distance metric function is defined and is used to find out the most similar set of tuples, $T_s$, with relation to the tuple to be classified, $t_q$. Considering the most frequent value of the target attribute in $T_s$, one can classify $t_q$. The most representative algorithm is called K-nn (K-nearest neighbour), where K is an input parameter. The most used distance metric functions are the Manhattan and Euclidean [20].

Using the SQL aware approach, there are two ways to determine the distances between the tuple to be classified and the tuples in the mining relation. The Beta primitive [7] allows the sorting of the tuples in the mining collection according to the similarity with $t_q$. Through the Compute Tuple Distance (CTD) [12] primitive, a DM application can determine the most similar tuple to $t_q$. There is a trade-off between these two solutions: while the Beta primitive is more general (allows to implement K-nn for $n > 1$), the CTD primitive is more efficient. The mapping of the Beta and CTD primitives are shown below in items (a) and (b), respectively.

(a) SELECT d($t_r$,$t_q$) AS d, [COUNT(*),] a_c FROM R [GROUP BY d, a_c] ORDER BY d
(b) SELECT a_c FROM R WHERE d($t_r$,$t_q$) IN (SELECT MIN(d($t_r$,$t_q$)) FROM R)

5 Conclusions

According to Sarawagi, Thomas and Agrawal [4], it is important to provide analysts with a well integrated platform where mining and database operations can be intermixed in flexible ways. However, many issues are involved in designing such platforms. In this work, we propose a classification of integration frameworks to help analysts in developing a DM application. We compared these frameworks under database coupling issues, such as portability, degree of coupling, flexibility and communication overhead. We believe that the tightly coupled framework has the best benefit/cost ratio over the conventional and the black box frameworks. The tightly coupled framework allows the DM application to get full advantage of the database system, while preserving development flexibility. We also show how current works are implementing the classification task benefiting from the several tightly coupled framework approaches.

References

SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada, 1996.


