Efficient Integration of Data Mining Techniques in Database Management Systems

Fadila Bentayeb Jérôme Darmont Cédric Udréa
ERIC, University of Lyon 2
5 avenue Pierre Mendès-France
69676 Bron Cedex
France
{bentayeb|jdarmont|cudrea}@eric.univ-lyon2.fr

Abstract

In this paper, we propose a new approach for applying data mining techniques, and more particularly supervised machine learning algorithms, to large databases, in acceptable response times. This goal is achieved by integrating these algorithms within a Database Management System. We are thus only limited by disk capacity, and not by available main memory. However, the disk accesses that are necessary to scan the database induce long response times. Hence, we propose an original method to reduce the size of the learning set by building its contingency table. The machine learning algorithms are then adapted to operate on this contingency table. In order to validate our approach, we implemented the ID3 decision tree construction method and showed that using the contingency table helped us obtaining response times equivalent to those of classical, in-memory software.

Keywords: *Databases, Data mining, Supervised machine learning, Decision trees, Contingency table, Performance.*

1. Introduction

The application of data mining operators to very large databases is an interesting challenge. However, data mining algorithms usually operate only on main memory data structures, more precisely on attribute-value tables. This limits the size of the processed databases. Traditional data mining approaches thus use techniques for preprocessing the data, such as feature selection [1] or sampling [2].

In order to apply data mining algorithms to large databases, several approaches have been proposed. They consist in integrating data mining methods in Database Management Systems (DBMSs) [3]. The first studies about integrating data analysis methods into DBMSs came with the development of data warehousing and On-Line Analysis Processing (OLAP) in particular [4]. Other related research efforts include the generation of association rules and their generalization [5, 6, 7]. However, few studies exist regarding the integration of classical data mining or analysis techniques such as clustering. In this domain, research indeed principally focuses on improving data mining methods for large databases [8]. Nevertheless, most DBMS vendors included data mining features into their products [9, 10, 11]. However, these integration attempts all took the form of "black boxes" requiring either extensions of the SQL language or the use of Application Programming Interfaces (APIs).

Hence, we proposed a different approach for integrating decision tree-like data mining methods, using only the tools offered by DBMSs [12]. More precisely, we implemented the ID3 method [13] within the Oracle DBMS as a PL/SQL stored procedure, by exploiting relational views. We showed that we could process very large databases with this approach, theoretically without any size limit, while classical, in-memory data mining software could not. However, processing times remained quite long because of multiple accesses to the database.

In order to improve these processing times, preparing the data before the data mining process becomes crucial. In this paper, we propose an original method to achieve this goal. We build a contingency table, i.e., a table that contains the frequencies and whose size is normally much smaller than the whole training set. The data mining methods are then adapted so that they can apply to this contingency table. To the best of our knowledge, no data mining method currently

uses such a data preparation phase.

To illustrate and validate our approach, we adapted our first implementation of ID3 and applied it to the contingency table obtained from a given training set. We show that using the contingency table allows clear improvements in terms of processing time, in comparison to our first, view-based implementation of ID3. Moreover, the processing times we achieve by using a contingency table are equivalent to those of classical, in-memory data mining software.

The remainder of this paper is organized as follows. Section 2 explains the principle of integrating decision tree-based methods within DBMSs. Section 3 details our contingency table-based decision tree construction method. Section 4 presents the experimental results and the complexity study that validate our proposal. Finally, Section 5 concludes this paper and presents research perspectives.

2. Integrating decision tree-based methods in DBMSs

2.1. Principle of decision trees

Decision trees are data mining models that produce "if-then"-like rules. They take as input a set of objects (tuples, in the relational databases vocabulary) described by a collection of variables (attributes). Each object belongs to one of a set of mutually exclusive classes. The induction task determines the class of any object from the values of its attributes. A training set of objects whose class is known is needed to build the tree. Hence, a decision tree building method takes as input a set of objects defined by predictive attributes and a class attribute, which is the attribute to predict.

These methods apply successive criteria on the training population to obtain a succession of smaller and smaller partitions of an initial training set, wherein the size of one class is maximized. This process builds a tree, or more generally a graph. Rules are then produced by following the paths from the root of the tree (whole population) to the different leaves (groups wherein the one class represents the majority in the frequency). Figure 1 provides an example of decision tree with its associated rules, where p(Class #i) is the probability of objects to belong to Class #i.

2.2. Decision tree modelling with relational views

In our first integrated approach [12], the key idea is to associate each node in the decision tree with its corresponding relational view. Hence, the root node of the decision tree is represented by a relational view corresponding to the whole training dataset. Since each sub-node in the decision tree represents a sub-population of its parent node, we build for each node a relational view that is based on its parent view. Then, these views are used to count the frequency of each class in the node with simple `GROUP BY` queries. These counts are used to determine the criteria that helps either partitioning the current node into a set of disjoint sub-partitions based on the values of a specific attribute or concluding that the node is a leaf, i.e., a terminal node. To illustrate how these views are created, we represent in Figure 2 the SQL statements for creating the views associated with the sample decision tree from Figure 1. This set of views constitutes the decision tree.

In order to validate this approach, we implemented the ID3 method within the Oracle DBMS as a PL/SQL stored procedure named $View\_ID3$. We showed that we could process very large databases with this approach, theoretically without any size limit, while classical, in-memory data mining software could not. We compared $View\_ID3$ to the in-memory ID3 implementation from the Sipina data mining software [14]. These results are presented in Figure 3, where the classical ID3 implementation is labelled $Sipina\_ID3$. Figure 3 shows that processing times remain quite long because of multiple accesses to the database. To reduce these processing times, we propose in this paper a new approach that uses a contingency table.

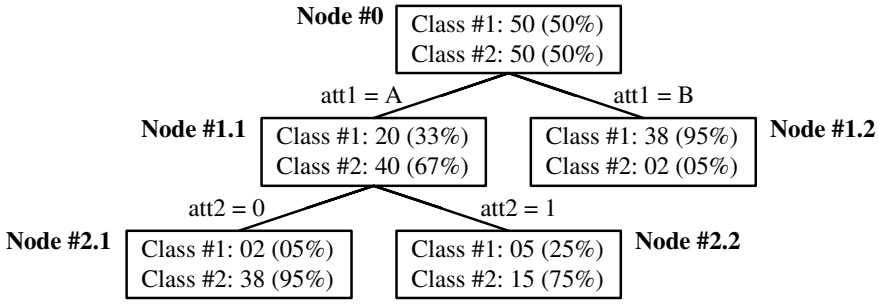
3. Contingency table-based decision tree construction

A contingency table is usually represented by means of a multidimensional table of frequencies that may contain NULL values. In our approach, since data mining algorithms are integrated within DBMSs and hence operate onto relational data structures, we represent contingency tables with relational tables or views that include an additional attribute to represent non NULL frequency values.

3.1. Construction of the contingency table

Within a DBMS, the contingency table corresponding to a given training dataset can be computed by a simple SQL query. Let TS be a training set defined by n predictive attributes $A_1, \ldots, A_n$ and the class attribute C. The associated contingency table CT is obtained by running the SQL query displayed in Figure 4.

Since the contingency table contains frequencies, its size is usually much smaller than the whole training



Rule #1: if att1 = A and att2 = 0 then p(Class #2) = 95%
Rule #2: if att1 = A and att2 = 1 then p(Class #2) = 75%
Rule #3: if att1 = B then p(Class #1) = 95%

Figure 1. Example of decision tree

```
Node #0:   CREATE VIEW v0 AS SELECT att1, att2, class FROM training_set
Node #1.1: CREATE VIEW v11 AS SELECT att2, class FROM v0 WHERE att1='A'
Node #1.2: CREATE VIEW v12 AS SELECT att2, class FROM v0 WHERE att1='B'
Node #2.1: CREATE VIEW v21 AS SELECT class FROM v11 WHERE att2=0
Node #2.2: CREATE VIEW v22 AS SELECT class FROM v11 WHERE att2=1
```

Figure 2. Relational views associated to the sample decision tree

set (and at most as large as the training set). Therefore, the gain in terms of processing time is normally significant.

3.2. Running example

To illustrate our approach, we use the TITANIC training set, which contains 2201 tuples defined by:

- three predictive attributes:
  - Class (1st, 2nd, 3rd, Crew),
  - Age (Adult, Child),
  - Gender (Male, Female);
- one class attribute: Survivor (Yes, No).

A sample from the TITANIC training set is provided in Table 1.

The classical contingency table corresponding to the TITANIC training set is provided in Table 2. Its relational representation is obtained with a simple SQL query (Figure 5). Its result contains only 24 tuples (Table 3).

3.3. Entropy and information gain

In the ID3 algorithm [13], the discriminating power of an attribute for segmentating of a node of the decision tree is expressed by a variation of entropy. The entropy h_s of a node s_k (more precisely, its entropy of Shannon) is:

$$h_s(s_k) = -\sum_{i=1}^{c} \frac{n_{ik}}{n_k} \log_2 \frac{n_{ik}}{n_k} \quad (1)$$

where n_k is the frequency of s_k and n_{ik} the number of objects of s_k that belong to class C_i. The information carried by a partition S_K of K nodes is then the weighted average of the entropies:

$$E(S_K) = \sum_{k=1}^{K} \frac{n_k}{n_j} h_s(s_k) \quad (2)$$

where n_j is the frequency of the segmented node s_j. Finally, the information gain associated to S_K is

$$G(S_K) = h_s(s_j) - E(S_K) \quad (3)$$

3.4. New formula for information gain

To show that our approach is efficient and pertinent, we adapted the ID3 method to apply to a contingency table. This induces changes for computing the information gain for each predictive attribute, and consequently for computing the entropy.

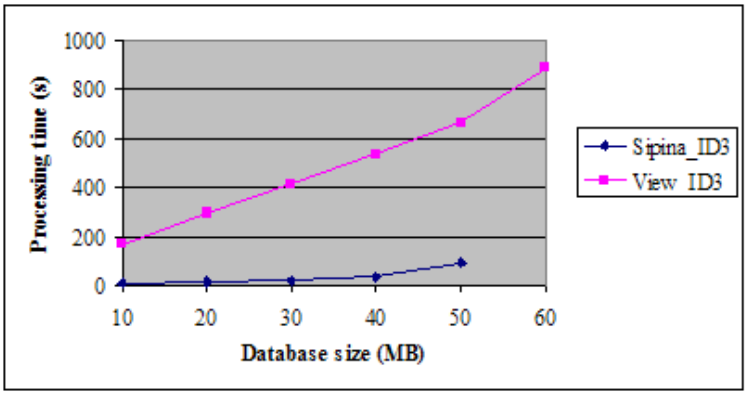


Figure 3. Performance comparison of $View\_ID3$ and $Sipina\_ID3$

```
CREATE VIEW CT_view AS
SELECT A1,...,An, C, COUNT(*) AS Frequency
FROM TS
GROUP BY A1,...,An, C
```

Figure 4. Relational view associated to contingency table CT

To compute the information gain for a predictive attribute, our view-based ID3 implementation reads all the tuples in the whole partition corresponding to the current node of the decision tree, in order to determine the tuple distribution regarding the values of each predictive attribute value and the class attribute. In our contingency table-based approach, it is quite simple to obtain the size of a subpopulation satisfying a given set of rules E_r (e.g., $Age = Child\ AND\ Gender = Female$) by summing the values of the Frequency attribute from the contingency table, for the tuples that satisfy E_r. Hence, we reduce the number of read operations to one only to calculate the information gain for a predictive attribute. Indeed, as presented in section 3.3, the usual calculation of the information gain for an attribute having k possible values and with a class attribute having c possible values is:

$$G(S_K) =$$

$$h_s(s_j) - \sum_{k=1}^{K} \left(\frac{n_k}{n_j} \times \left(- \sum_{i=1}^{c} \frac{n_{ik}}{n_k} \times log_2 \left(\frac{n_{ik}}{n_k} \right) \right) \right) \quad (4)$$

where n_j is the node frequency, n_k is the frequency of the subnode having value V_k for the predictive attribute, n_{ik} is the frequency of the subnode partition having value V_k for the predictive attribute and value C_i for the class attribute. However, if we develop Formula 4, and since $log_2 \frac{a}{b} = log_2 a - log_2 b$, by adding up n_{ik} and n_k, we obtain:

$$G(S_K) =$$

$$h_s(s_j) + \frac{1}{n_j} \times \left(\sum_{k=1}^{K} \sum_{i=1}^{c} n_{ik} \times log_2 n_{ik} - \sum_{k=1}^{K} n_k \times log_2 n_k \right) \quad (5)$$

By applying Formula 5 to the contingency table (that we read only once), we obtain the information gain easily. Indeed, in this formula, it is not necessary to know at the same time various frequency (n_j, n_k, n_{ik}), and we obtain n_k by summing the n_{ik} and n_j by summing the n_k.

4. Validation

4.1. Implementation

We use Oracle to implement our adaptation of ID3 to contingency tables, under the form of a PL/SQL stored procedure named CT_ID3, which is part of a broader package named *decision_tree* that is available on-line[1]. We have tested this implementation under Oracle version 8i and 9i.

1 http://bdd.univ-lyon2.fr/download/decision_tree.zip

Class	Age	Gender	Survivor
1st	Adult	Male	Yes
2nd	Adult	Male	No
Crew	Adult	Female	Yes
3rd	Child	Male	Yes
1st	Adult	Female	No
1st	Child	Female	Yes
Crew	Adult	Male	Yes
2nd	Child	Male	Yes
Crew	Adult	Female	Yes
3rd	Adult	Female	No
1st	Child	Male	Yes
2nd	Adult	Female	No
2nd	Adult	Male	Yes
3rd	Adult	Male	No
1st	Child	Female	Yes
2nd	Child	Female	Yes
2nd	Adult	Female	No
1st	Adult	Male	No
3rd	Child	Female	Yes
Crew	Adult	Female	Yes
3rd	Adult	Male	No
2nd	Child	Female	Yes
Crew	Adult	Male	No
Crew	Adult	Female	No
1st	Child	Female	Yes
3rd	Adult	Male	No
...	...	...	...

Table 1. TITANIC training set sample

```
CREATE VIEW TITANIC_Contingency AS
SELECT Class, Gender, Age, Survivor, COUNT(*) AS Frequency
FROM TITANIC
GROUP BY Class, Gender, Age, Survivor
```

Figure 5. Relational view associated to the TITANIC contingency table

4.2. Experimental results

In order to validate our contingency table-based approach and to compare its performances to those of the previous approaches, we carried out tests on different views of the CovType database[2]. The CovType database contains 581 012 tuples defined by 54 predictive attributes and one class (with seven different values). We created five views containing each a part of the Covtype database defined by three predictive attributes (each has five different values) and the class. The predictive attributes used and the size for each view are provided in Table 4.

First, we compare the processing times of our ID3 implementations (the view-based approach $View\_ID3$ and the contingency table-based approach CT_ID3). These tests have been carried out on a PC computer with 128 MB of RAM and the Personal Oracle DBMS version 9i. The use of Personal Oracle ensures that the DMBS client and server were on the same machine, hence no network traffic can interfere with our per-

2 http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/

		Adult		Child	
		Male	Female	Male	Female
1st	Yes	57	140	5	1
	No	118	4	0	0
2nd	Yes	14	80	11	13
	No	154	13	0	0
3rd	Yes	75	76	13	14
	No	387	89	35	17
Crew	Yes	192	20	0	0
	No	670	3	0	0

Table 2. Classical contingency table for TITANIC

Class	Age	Gender	Survivor	Frequency
1st	Adult	Male	Yes	57
1st	Adult	Male	No	118
1st	Adult	Female	Yes	140
1st	Adult	Female	No	4
1st	Child	Male	Yes	5
1st	Child	Female	Yes	1
2nd	Adult	Male	Yes	14
2nd	Adult	Male	No	154
2nd	Adult	Female	Yes	80
2nd	Adult	Female	No	13
2nd	Child	Male	Yes	11
2nd	Child	Female	Yes	13
3rd	Adult	Male	Yes	75
3rd	Adult	Male	No	387
3rd	Adult	Female	Yes	76
3rd	Adult	Female	No	89
3rd	Child	Male	Yes	13
3rd	Child	Male	No	35
3rd	Child	Female	Yes	14
3rd	Child	Female	No	17
Crew	Adult	Male	Yes	192
Crew	Adult	Male	No	670
Crew	Adult	Female	Yes	20
Crew	Adult	Female	No	3

Table 3. Relational representation of the TITANIC contingency table

formance measurements. The results we obtain (Figure 6) clearly underline the gain induced by CT_ID3, that has a much lower processing time than $View\_ID3$ on an average, while producing the same result.

Then, we compare the processing times of CT_ID3 to the ID3 implementation available in Sipina ($Sipina\_ID3$). On small databases, we observe that our contingency table-based approach presentes similar processing times than $Sipina\_ID3$. Furthermore it can also process larger databases while $Sipina\_ID3$ remains limited by main memory (Figure 7).

The processing time of our approach depends mainly on the size of the contingency table. In conclusion, we hint the effectiveness of our approach, since it generally considerably reduces the size of the training set. Thus, the use of a contingency table as an optimization tool within the framework of integrating data mining methods in DBMSs improves processing times sig-

View name	Predictive attributes used	View size	Size of contingency table
view1	1,2,3	116202 tuples	322 tuples
view2	4,5,6	232404 tuples	265 tuples
view3	7,8,9	348607 tuples	214 tuples
view4	1,4,10	464810 tuples	202 tuples
view5	2,5,8	581012 tuples	264 tuples

Table 4. Views used in CovType tests

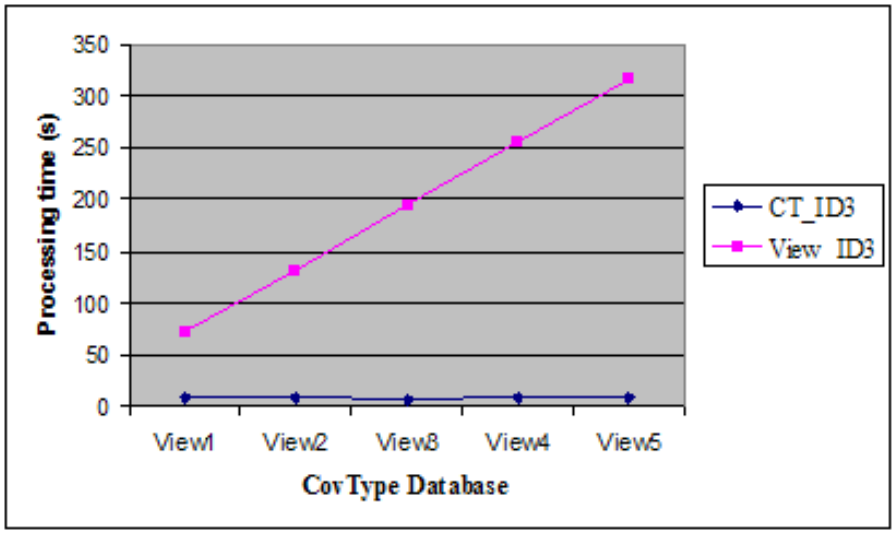


Figure 6. Performance comparison of CT_ID3 and $View\_ID3$

nificantly. Nevertheless, in extreme cases, the size of the contingency table may be so close to that of the whole training set that the profit becomes negligible. However, this is very rare in real-life cases and scanning the contingency table can never be worse than scanning the whole database.

4.3. Complexity study

Our objective is to compare the complexity of our both integrated approaches (CT_ID3 and $View\_ID3$) in terms of processing time. We suppose that both algorithms are optimized in their implementation so that only the necessary tuples are read. In this study, we are interested in the time spent reading and writing data, since these are the most expensive operations. We consider that a tuple is read or written in one time unit. Finally, we consider that the obtained decision tree is balanced and whole, i.e., that at each level of the tree, the union of the populations of the various nodes equals the whole database.

Let N be the total number of tuples in the training set. Let K be the number of predictive attributes. Let T be the size of the corresponding contingency table.

With $View\_ID3$, to reach level $i+1$ from an unspecified level i of the tree, each node must be read as many times as there are predictive attributes at this level, i.e., $(K-i)$. As the sum of the frequencies at this level corresponds to the frequency of the starting database, it is thus necessary to read N tuples $(K-i)$ times (number of tuples $\times$ size of a tuple $\times$ number of attributes). Hence, the total reading time for level i is $N(K-i)$. In order to reach this level, it is also necessary to write the corresponding tuples. The writing time is thus N.

Since $\sum_{i=1}^{K} i = K(K+1)/2$, we obtain the following final complexity, from the root to the leaves (level K):

- reading complexity: $N(K^2/2 - K/2)$ time units, therefore NK^2;
- writing complexity: NK time units.

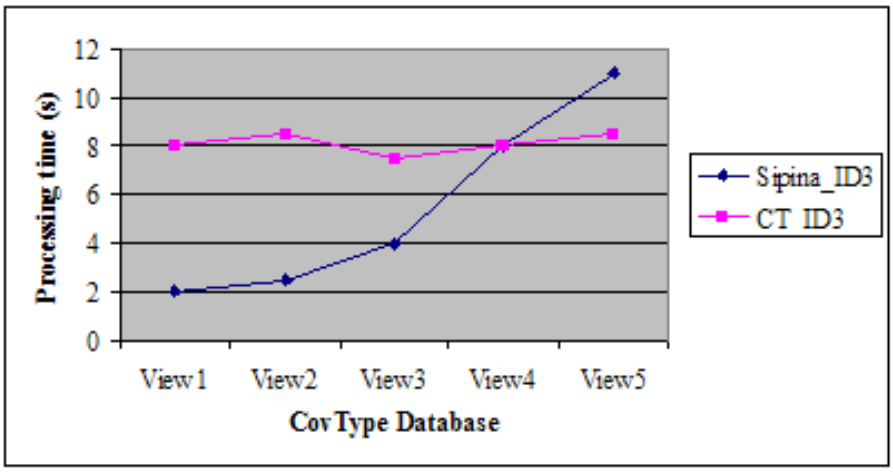


Figure 7. Performance comparison of CT_ID3 and $Sipina\_ID3$

In our contingency table-based approach, we first create the contingency table the writing time is thus T. To compute the contingency table, we read the whole database once. The reading time is thus N. When reaching level $i+1$ from level i, we read all the T tuples $(K-i)$ times, for a total time by level of $T(K-i)$.

Hence, with CT_ID3, the complexity results are:

- reading complexity: $T(K^2/2 - K/2) + N$ time units, therefore TK^2 or N if $N > TK^2$;
- writing complexity: T time units.

In conclusion, in terms of processing time, our contingency table-based approach allows an improvement of N/T or K^2 (if $N > TK^2$) for reading, and of NK/T for writing. Since N is usually much greater than T, this improvement is significant.

5. Conclusion and Perspectives

In order to apply data mining algorithms to large databases, two main approaches are proposed in the literature: the classical approach and the integrated approach. The classical approach is limited by the size of the processed databases since it operates in main memory. The main objective in this approach is then to reduce the size of databases, either by using techniques for preprocessing data or by sampling. The integrated approach consists in processing data mining methods within DBMSs using only the tools offered by these systems. By exploiting their management of persistent data, the database size limit is toppled. Our first contribution in this domain was the integration of the ID3 method within Oracle by means of relational views associated to the nodes of a decision tree ($View\_ID3$) [12]. However, creating and exploiting views generates multiple accesses to the database, and hence long processing times.

Following the integrated approach, we proposed in this paper an original method to apply data mining algorithms by taking into account not only the size of processed databases, but also processing time. Our key idea consists in replacing the whole training set by its contingency table represented by means of a relational view. Our approach presents two advantages. First, it is easy to build this contingency table with a simple SQL query. Second, the size of the contingency table is much smaller than the whole training set in most practical cases.

To validate our approach, we implemented the ID3 method as a PL/SQL stored procedure named CT_ID3, and showed that processing times were greatly improved in comparison to $View\_ID3$. We also showed that the performances of CT_ID3 were comparable to those of a classical, in-memory implementation of ID3.

The perspectives opened by this study are numerous. In order to enrich our *decision_tree* software package, we are currently implementing other data mining methods, such as C4.5 and CART, using contingency tables. Moreover, we plan to compare the performances of these implementations to their equivalent in the Sipina software (that operate in memory) on real-life databases. Moreover, we plan to add in the *decision_tree* package other procedures to supplement the offered data mining tools, such as sampling, missing values management, and learning validation tech-

niques.

Finally, to improve their processing time, in-memory data mining methods could also use contingency tables instead of original learning sets.

References

[1] H. Lia and H. Motoda, *Feature Selection for knowledge discovery and data mining.* Kluwer Academic Publishers, 1998.

[2] J. Chauchat and R. Rakotomalala, "A new sampling strategy for building decision trees from large databases," in *7th Conference of the International Federation of Classification Societies (IFCS 00), Namur, Belgium*, 2000, pp. 199–204.

[3] S. Chaudhuri, "Data mining and database systems: Where is the intersection?" *Data Engineering Bulletin*, vol. 21, no. 1, pp. 4–8, 1998.

[4] E. F. Codd, "Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate," E.F. Codd and Associates, Tech. Rep., 1993.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo, "Fast discovery of association rules," in *Advances in Kowledge Discovery and Data Mining*, 1996, pp. 307–328.

[6] R. Meo, G. Psaila, and S. Ceri, "A new SQL-like operator for mining association rules," in *22th International Conference on Very Large Data Bases (VLDB 96), Mumbai, India*, 1996, pp. 122–133.

[7] S. Sarawagi, S. Thomas, and R. Agrawal, "Integrating mining with relational database systems: Alternatives and implications," in *ACM SIGMOD International Conference on Management of Data (SIGMOD 98), Seattle, USA*, 1998, pp. 343–354.

[8] J. Gehrke, R. Ramakrishnan, and V. Ganti, "Rainforest - a framework for fast decision tree construction of large datasets," in *24th International Conference on Very Large Data Bases (VLDB 98), New York City, USA*, 1998, pp. 416–427.

[9] IBM, "DB2 intelligent miner scoring," http://www-4.ibm.com/software/data/iminer/scoring, 2001.

[10] Oracle, "Oracle 9i data mining," White paper, June 2001.

[11] S. Soni, Z. Tang, and J. Yang, "Performance study of Microsoft data mining algorithms," Microsoft Corp., Tech. Rep., 2001.

[12] F. Bentayeb and J. Darmont, "Decision tree modeling with relational views," in *XIIIth International Symposium on Methodologies for Intelligent Systems (ISMIS 02), Lyon, France*, ser. LNAI, vol. 2366, June 2002, pp. 423–431.

[13] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[14] R. Rakotomalala, "Sipina research," http://eric.univ-lyon2.fr/∼ricco/sipina.html, 2003.