Efficient online mining of large databases

Fadila Bentayeb, Jérôme Darmont and Cécile Favre*

ERIC, University of Lyon 2,
5 avenue Pierre Mendès-France,
69676 Bron Cedex, France
E-mail: bentayeb@eric.univ-lyon2.fr
E-mail: jerome.darmont@univ-lyon2.fr
E-mail: cfavre@eric.univ-lyon2.fr
*Corresponding author

Cédric Udréa

EURISE, University of St Etienne,
23 rue du Docteur Paul Michelon,
42023 Saint Etienne Cedex 2, France
E-mail: cedric.udrea@univ-st-etienne.fr

Abstract: Great efforts have been achieved to apply data mining algorithms onto large databases. However, long processing times remain a practical issue. This paper presents a framework to offer to database users online operators for mining large databases without size limit, in acceptable processing times. First, we integrate decision tree algorithms directly into database management systems. We are thus only limited by disc capacity and not by main memory. However, disc accesses still induce long response times. Hence, we propose two optimisations in a second step: reducing the size of the learning database by building its corresponding contingency table and reducing the number of database accesses by exploiting bitmap indices. Thus, the various decision tree based methods we implemented within Oracle deal with contingency tables or bitmap indices rather than with the whole training set. Experimentations performed show the efficiency of our integrated methods.

Keywords: bitmap indices; contingency table; databases; decision trees; online data mining; performance; relational views.

Reference to this paper should be made as follows: Bentayeb, F., Darmont, J., Favre, C. and Udréa, C. (2007) 'Efficient online mining of large databases', *Int. J. Business Information Systems*, Vol. 2, No. 3, pp.328–350.

Biographical notes: Fadila Bentayeb has been an Associate Professor at the University of Lyon 2, France, since 2001. She is a Member of the Decision Support Databases Research Group within the ERIC laboratory. She received her PhD degree in Computer Science from the University of Orléans, France, in 1998. Her current research interests regard database management systems, including the integration of data mining techniques into DBMSs and data warehouse design, with a special interest for schema evolution, XML and complex data warehousing, benchmarking and optimisation techniques.

Jérôme Darmont received his PhD degree in Computer Science from the University of Clermont-Ferrand II, France, in 1999. He has been an Associate Professor at the University of Lyon 2, France, since then, and became the Head of the Decision Support Databases Research Group within the ERIC laboratory in 2000. His current research interests mainly relate to the evaluation and optimisation of database management systems and data warehouses (benchmarking, auto-administration, optimisation techniques, etc.), but also include XML and complex data warehousing and mining, and medical or health-related applications.

Cécile Favre is currently a PhD student in Computer Science at the University of Lyon 2, France. She is a Member of the Decision Support Databases Research Group within the ERIC laboratory. She received her MSc degree in Knowledge Discovery in Databases from the University of Lyon 2 in 2003. After researching on the integration of data mining techniques into DBMSs, her current research interests now relate to data warehouse design and evolution, and especially the integration of users' knowledge in data warehouses.

Cédric Udréa received his MSc degree in Knowledge Discovery in Databases from the University of Lyon 2, France, in 2004. His Masters thesis was about integrating data mining techniques into DBMSs. He is currently a PhD student in Computer Science at the University of St Etienne, France, and the Member of the EURISE laboratory. There, he is working on pattern database management in the context of the BINGO project (INductive dataBases and GEnomics in English).

1 Introduction

The input of traditional data mining algorithms are data structured as attribute-value tables. Since these algorithms operate in main memory, the size of the processed databases is limited. Nowadays, one of the key challenges in Knowledge Discovery in Databases (KDD) is to integrate data mining methods within the framework of traditional database systems so that their implementations can take advantage of the efficiency provided by SQL engines (Chaudhuri, 1998).

Data mining and databases should indeed not remain as separate components in decision support systems. Integrating data mining tools into Database Management Systems (DBMSs) is a promising research direction for the following reasons.

- Data mining tools need integrated, consistent and cleaned data. A database is precisely constructed through such pre-processing steps.
- Data mining algorithms operate in main memory, which limits the size of the processed databases. DBMSs provide a framework to manage large databases without size limit, theoretically.
- Some of the more popular data mining algorithms, namely decision tree methods, compute many successive frequencies to build trees. The SQL language includes COUNT and GROUP BY commands to easily compute such frequencies. Moreover, the use of indices can improve data access time when processing the database.

- When data warehouses have been stored into relational databases, OnLine Analytical Processing (OLAP) has been integrated within DBMSs. In the same way, we propose to extend DBMSs' analysis features with online data mining tools.

In this paper, we propose a full integrated solution for mining large databases within DBMSs. We aim at the following two main objectives: mine very large databases without size limit and achieve acceptable processing times. Moreover, in opposition to the integrated approaches proposed in the literature, our approach also presents two main advantages: no extension of the SQL language is needed and no programming through an Application Programming Interface (API) is required. We achieve our first objective by integrating data mining algorithms, especially decision tree-based methods, within DBMSs. However, processing times are quite long. To improve processing time and carry out our second objective, we efficiently exploit some structures and primitives provided by SQL engines for data retrieval.

In our approach, we propose three integrated methods: a view-based method (Bentayeb and Darmont, 2002), a contingency table-based method (Bentayeb, Darmont and Udréa, 2004) and a bitmap index-based method (Favre and Bentayeb, 2005). Each method is based on a specific database tool.

1. *View-based method*. Decision tree methods generate a tree (or more generally a graph) that is a succession of smaller and smaller partitions of an initial training set (table or view). Our key idea comes from this very definition. Indeed, we can make an analogy between building successive, related partitions and creating successive, related relational views. Each node of the decision tree is then associated to the corresponding view. Then, to build thd decision tree, we only need relational views that we exploit through SQL queries. We show that we can process very large databases with this method, theoretically without any size limit, while classical, in-memory data mining software cannot. However, processing times are quite long because of multiple accesses to the database.
2. *Contingency table-based method*. In order to improve processing times, preparing the data before the data mining process becomes crucial. We propose an original method to achieve this goal, which comprises reducing the size of the training set. We build a contingency table, i.e. a table that contains the frequencies, corresponding to the whole training set and whose size is normally much smaller than the table containing the whole training set. 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.
3. *Bitmap index-based method*. Another method for improving processing times comprises reducing the number of data accesses within the DBMS. The method we propose exploits database indices, namely bitmap indices that have many useful properties, such as count and bitwise operations that we exploit through SQL queries to build decision trees. Our method presents an important advantage because there is no need to access the source data, since we deal with bitmap indices rather than with the whole training set.

We implemented different decision tree algorithms, such as ID3, C4.5 and CART following our three methods within the Oracle DBMS, as PL/SQL stored procedures. In this paper, we detail the algorithm and performance results for the ID3 method, which are quite similar than those of C4.5 and CART. We observe that our integrated approach allows to process larger databases than in-memory implementations while presenting interesting processing times.

This paper expands our previous work along four axes. First, our motivation in this paper is to globally present our integrated approach as a whole. Second, we present a complete overview of the existing approaches for mining large databases from both the data mining and the database fields and compare them to our solution. Third, we detail implementation issues. Finally, we present new experiments on several data sets and discuss the results we obtained when comparing our three integrated methods.

The remainder of this paper is organised as follows. First, we discuss the related work regarding large databases mining in Section 2. Section 3 presents the principles of decision tree-based methods. In Sections 4, 5 and 6, we detail our different integrated methods and present their implementation, as well as complexity studies, respectively. We also present, in Section 7, the experiments we performed to validate our approach. We finally conclude this paper and discuss the research perspectives in Section 8.

2 Related work

Efficiently mining large databases has been the subject of many research studies for several years. Since traditional data mining algorithms operate in main memory, the size of the processed databases is limited. Different approaches have emerged to overcome this limit. The first one comprises pre-processing of the data to reduce the size of the learning databases. The second one uses optimisation techniques to assure the methods' scalability. The third one develops tools for integrating data mining algorithms into DBMSs.

2.1 Data pre-processing

Variable and feature selection have become the focus of much research in areas of application for which datasets with tens or hundreds of thousands of variables are available. The objective of variable selection is to improve the prediction performance of the predictors. In fact, it comprises exploiting the data pre-processing techniques. First, feature selection (Liu and Motoda, 1998; Fu and Wang, 2005) aims at reducing the number of predictive attributes. The feature selection must assume that the attributes that are deleted from the learning population do not impact the learning result, i.e. it must delete the less-pertinent attributes for learning. Sampling techniques (Toivonen, 1996; Chauchat and Rakotomalala, 2001; Scheffer and Wrobel, 2002) aim at considering fewer individuals for learning. The main objective is to obtain a sampling of the learning population that is representative of the whole population. However, the learning quality must not be decreased. It has indeed been proved that, with a well-chosen sampling, decision tree algorithms can provide better results than with the whole learning population (Chauchat and Rakotomalala, 2001).

2.2 Scalability

Data mining often induces problems of combinatorial explosion in terms of space and time. Thus, there has been an impressive amount of work related to scalability, which focuses on scaling data mining techniques to work on large datasets. Scalability is achieved by two means:

- Optimising the algorithms (Agrawal et al., 1996; Gehrke, Ramakrishnan and Ganti, 2000; Lee, Park and Park, 2003), i.e. exploring how to improve the efficiency of the mining algorithms.
- Optimising data accesses (Dunkel and Soparkar, 1999; Lu and Liu, 2000; Ramesh, Maniatty and Zaki, 2002), i.e. focusing on the impact of representation, organisation and access to data on performance of mining algorithms.

2.3 Integrated methods

Recently, a new approach has emerged to apply data mining algorithms on large databases. It comprises integrating data mining methods within DBMSs (Chaudhuri, 1998). A first step in this integration process has been achieved with the rise of data warehousing, whose primary purpose is decision support rather than reliable storage. A closely related area is OLAP (Codd, 1993). Database vendors also recently integrated data mining methods into their systems under the form of ‘black box’ tools, either by developing extensions of SQL or by developing ad hoc APIs (Oracle, 2001; Soni, Tang and Yang, 2001). These tools allow client applications to explore and manipulate the existing mining models and their applications through an interface similar to that used for exploring tables, views and other first-class relational objects.

Many other integrated approaches have been proposed in the literature. They usually use either extensions of SQL for developing new operators (Meo, Psaila and Ceri, 1998; Sarawagi, Thomas and Agrawal, 1998; Geist and Sattler, 2002), new languages (Han et al., 1996; Imielinski and Virmani, 1999; Elfeky, Saad and Fouad, 2000; Wang, Zaniolo and Luo, 2003; Meo, 2003; Feng and Dillon, 2005; Luo et al., 2005) or extensions of the DBMS itself by introducing the concept of ‘virtual mining view’ (Calders, Goethals and Prado, 2006).

There are also advances in the context of integrated approaches that neither use any API nor extensions of SQL. A new index type has indeed been proposed (Morzy and Zakrzewicz, 1998). It can be considered as an extension of bitmap indices and helps improving subset searching in large databases. This approach could be used in the field of association rule mining. Moreover, in Ordonez (2006), the author proposes to integrate the K-Means clustering method with a relational DBMS using SQL.

In conclusion, integrating data mining algorithms within the framework of traditional database systems becomes one of the key challenges for research in both the database and the data mining fields (Chaudhuri, 1998). Indeed, it provides online data mining operators to the users in addition to the usual SQL operators.

3 Decision tree-based methods

3.1 Principle

Decision trees are among the most popular supervised learning methods proposed in the literature. They are appreciated for their simplicity and the high efficiency of their algorithms, for their ease of use and for the easily interpretable results they provide.

Many induction tree methods have been proposed so far in the literature. Some, such as 'Induction Decision Tree' (ID3) (Quinlan, 1986) and C4.5 (Quinlan, 1993), build *n*-ary trees. Others such as 'Classification And Regression Tree' (CART) (Breiman et al., 1984) produce binary trees.

An induction tree may be viewed as a succession of smaller and smaller partitions of an initial training set. It takes a set of objects (tuples) described by a collection of attributes as the input. 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 induction graph. Hence, an induction graph building method takes a set of objects defined by predictive attributes and a class attribute, which is the attribute to predict as the input.

Decision tree construction methods apply successive criteria on the training population to obtain these partitions, wherein the size of one class is maximised. In the ID3 algorithm, for example, the discriminating power of an attribute for splitting 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

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

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

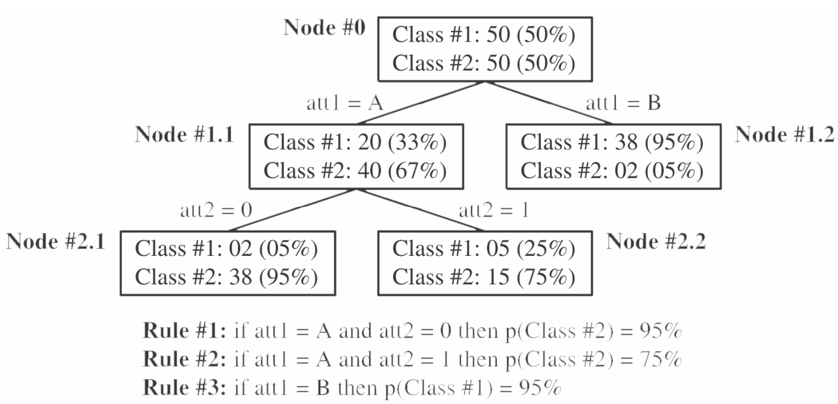
$$E(S_K) = \sum_{k=1}^{K} \frac{n_k}{n_j} h_s(s_k) \tag{2}$$

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

$$G(S_K) = h_s(s_j) - E(S_K) \tag{3}$$

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.

Figure 1 Example of decision tree



3.2 Running example

To illustrate how the different methods presented in this paper operate, we use the TITANIC database as an example (Table 1), which is a training set of 2201 tuples. It is commonly used to test decision tree building methods.

Table 1 TITANIC database

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

The aim is to predict which classes of passengers of the TITANIC are more likely to survive the wreck. Those passengers are described by three predictive attributes: *Class* = {*lst*; *2nd*; *3rd*; *Crew*}; *Age* = {*Adult*; *Child*}; *Gender* = {*Female*; *Male*} and the attribute to predict *Survivor* = {*No*; *Yes*}.

4 View-based method

4.1 Principle

In our first integrated method (Bentayeb and Darmont, 2002), the key idea is to associate each node in the decision tree with its corresponding relational view. In this method, 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 a relational view for each node which 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 criterion 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 our method, we show in Figure 2 how these views are created based on the TITANIC training set (Table 1). Then we represent in Figure 3 the SQL statements for creating the views associated to the sample decision tree from Figure 2. This set of views constitutes the decision tree.

Figure 2 TITANIC sample decision tree

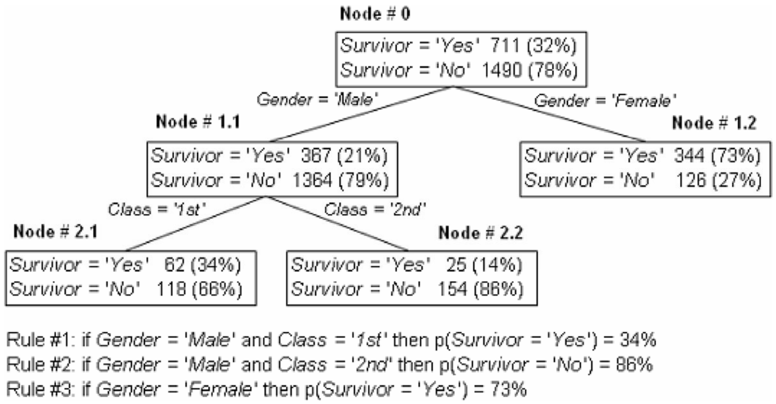


Figure 3 Relational views associated with the TITANIC sample decision tree

```
Node #0:    CREATE VIEW v0 AS SELECT Age, Gender,
Class,Survivor
               FROM TITANIC
Node #1.1:  CREATE VIEW v11 AS SELECT Age, Class, Survivor FROM
              v0 WHERE Gender='Male'
Node #1.2:  CREATE VIEW v12 AS SELECT Age, Class, Survivor FROM
              v0 WHERE Gender='Female'
Node #2.1:  CREATE VIEW v21 AS SELECT Age, Survivor FROM v11
              WHERE Class='1st'
Node #2.2:  CREATE VIEW v22 AS SELECT Age, Survivor FROM v11
              WHERE Class='2nd'
```

4.2 Implementation

We present the algorithm for the ID3 method that we call *View_ID*3. We implemented this algorithm within the Oracle 10 g DBMS as a PL/SQL stored procedure.

Algorithm

Input parameters. The input parameters of our algorithm are given in Table 2.

Table 2 *View_ID*3 algorithm input parameters

Parameter	*Name*	*Default value*
Data source table name	table_name	–
Class attribute (attribute to predict)	class	–
Results table name	res_name	BTRES
(Strict) minimum information gain for node building	min_gain	0
Root node view name	root_view	BTROOT
Clean-up views after execution (True/False)	del	TRUE

Pseudo-code. We call a procedure named Entropy () that computes both the entropy and the frequency of a node. These data are used when computing the information gain. Entropy () is coded in PL/SQL. Our algorithm pseudo-code for the *View_ID*3 procedure is provided in Figure 4.

Figure 4 Pseudo-code for *View_ID*3 stored procedure

```
Create result table
Create root node using the data source table
Compute root node entropy and frequency
Push root node
Update result table with root node
While the stack is not empty do
    Pop current node
    Clean candidate list
    For each attribute but the class attribute do
       Create a new candidate
       For each possible value of current attribute do
          Build new node and associated relational view
          Compute new node entropy and frequency
          Update information gain
          Insert new node into current candidate node list
       End for (each value)
    End for (each attribute)
    Search for maximum information gain in candidate list
    For each candidate do
       If current attribute bears the greater information
gain then
          For each node in the list of nodes do
             Push current node
             Update result table with current node
          End for (each node)
       Else
          For each node in the list of nodes do
             Destroy current node
          End for (each node)
       End if
    End for (each candidate)
End while (stack not empty)
```

Result output. The output of our stored procedure, namely a decision tree, is stored into a relational table whose name is specified as an input parameter. The table structure reflects the hierarchical structure of the tree. Its fields are:

- node, the node ID number (primary key, the root node ID is always #0 – note that there is a direct link between the node ID and the associated view name).
- parent, the ID number of the parent node in the tree (foreign key, references a node ID number).

- rule, the rule that lead to the creation of this node, e.g. *Gender* = '*Female*'.
- frequency, for each value V of attribute E, a field labelled E_V, the frequency for the considered value of the attribute in this node.

Such a table is best queried using Oracle SQL hierarchical statements. The result is directly a textual description of the output decision tree. A sample query is provided in Figure 5. From this representation, it is very easy to deduce the corresponding set of production rules.

Figure 5 Hierarchical SQL query for decision tree display

```
SELECT LEVEL, node, parent, rule, E_1, E_2, ...
FROM btres
CONNECT BY node = parent START WITH node = 0
```

5 Contingency table-based method

5.1 Definition

A contingency table is usually represented by means of a multidimensional table of frequencies that may contain NULL values. In our approach, data mining algorithms are integrated within DBMSs and hence operate onto relational data structures. In this context, contingency tables are represented by means of relational tables or views and contain only non-NULL frequency values. This considerably reduces the size of the table. An additional attribute is then added to the contingency table structure to represent frequency values.

5.2 Principle

In this method (Bentayeb et al., 2004), we aim at reducing the size of the initial training set to improve processing times. Thus, we build the contingency table, i.e. a table that contains the frequencies corresponding to the whole training set. It can be computed by a simple SQL query. For example, 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 executing the SQL query displayed in Figure 6.

Figure 6 Relational view associated to contingency table *CT*

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

Therefore, decision tree methods have to be adapted to be applied on this contingency table whose size is normally much smaller than the initial training set. Hence, the gain in terms of processing time is normally significant.

5.3 Running example and implementation

The classical contingency table corresponding to the TITANIC training set (Table 1) is provided in Figure 7. Its relational representation is obtained with a simple SQL query (Figure 8). Its result contains only 24 tuples (Figure 9).

Figure 7 Classical contingency table for TITANIC

		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

Figure 8 Relational view associated to the TITANIC contingency table

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

Figure 9 Relational representation of the TITANIC contingency table

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

We used Oracle 10g to implement our adaptation of ID3 to contingency tables under the form of a PL/SQL stored procedure named *CT_ID*3.

5.4 New formula for the information gain

Since the training set is a contingency table (a table containing frequencies), this induces changes for computing the information gain for each predictive attribute and, consequently, for computing the entropy.

To compute the information gain for a predictive attribute, our view-based ID3 implementation (*View_ID*3) 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 and the class attribute. In our contingency table-based method, it is quite simple to obtain the size of a sub-population 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 for computing the information gain of a predictive attribute. Indeed, as presented in Section 3, the usual calculation of the information gain for an attribute having k possible values and with a class attribute having c possible value is

$$G(S_K) = h_s(s_j) - \sum_{k=1}^{K} \left\{ \frac{n_k}{n_j} \times \left[-\sum_{i=1}^{c} \frac{n_{\text{ik}}}{n_k} \times \log_2 \left(\frac{n_{\text{ik}}}{n_k} \right) \right] \right\} \tag{4}$$

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

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

By applying Equation (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 various frequencies at the same time (n_j, n_k, n_{ik}), and we obtain n_k by summing the n_{ik} and n_j by summing the n_k.

5.5 Complexity study

Our objective here is to compare the complexity of both our integrated methods (*CT_ID*3 and *View_ID*3) in terms of processing times. We suppose that both algorithms are optimised 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. 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_ID*3, 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 × size of a tuple × 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.

In our contingency table-based method, 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_ID*3, 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 times, our contingency table-based method 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.

6 Bitmap index-based method

6.1 Principle

Bitmap indices improve the performance of SQL queries including COUNT or bitwise operations by not accessing the source data. This type of queries is similar to those we need to build a decision tree and more precisely to define the size of the nodes' sub-populations. Indeed, as we are going to explain next, in Table 4, to find the total number of '*male survivors*', the SQL engine performs logical AND and COUNT operators onto bitmap indices and retrieves the result without accessing the source data. In the case of a decision tree-based method, this query may correspond to a splitting step for obtaining the frequency of class *Survivor* = '*Yes*' in the node corresponding to the rule *Gender* = '*Male*'. Our key idea comes from this very definition (Favre and Bentayeb, 2005).

6.2 Bitmap indices

Originally, a bitmap index is a data structure used to efficiently access large databases (O'Neil, 1987; O'Neil and Quass, 1997). Generally, the purpose of an index is to provide pointers to the rows in a table that contain a given key value. In a regular index, this is achieved by storing a list of row identifiers (RowIds) for each key corresponding to the rows with that key value. In a bitmap index, records in a table are assumed to be numbered sequentially from 1. For each key value, a bitmap (array of bits) is used instead of a list of RowIds. Each bit in the bitmap corresponds to a possible RowId. If the bit is set to '1', the row with the corresponding RowId contains the key value; otherwise, the bit is set to '0'. A mapping function converts the bit position to an actual RowId; hence the bitmap index provides the same functionality as a regular index even though it internally uses a different representation.

Example. To illustrate how bitmap indices work, we use as an example, the TITANIC database, presented in Table 1. A bitmap index on the *Survivor* attribute is presented in Table 3.

Table 3 *Survivor* bitmap index

	RowId	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivor	No	..	1	1	1	0	1	0	1	0	0	0	0	0
	Yes	..	0	0	0	1	0	1	0	1	1	1	1	1

Table 4 Bitmap (*Survivor* = '*Yes*') **AND** bitmap (*Gender* = '*Male*')

RowId	..	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivor = '*Yes*'	..	0	0	0	1	0	1	0	1	1	1	1	1
Gender = '*Male*'	..	1	1	1	0	0	1	1	0	1	1	1	0
AND	..	0	0	0	0	0	**1**	0	0	**1**	**1**	**1**	0

Properties. Bitmap indices are designed for efficient queries on multiple keys. Hence, queries are answered using bitwise operations such as intersection (AND) and union (OR). Each operation exploits two bitmaps of the same size and is applied on corresponding bits to get the result bitmap. Every '1' bit in the result marks the desired tuple. Counting the number of tuples in the result is even faster. For queries such as 'SELECT COUNT () … WHERE … AND … OR …', the logical operations can provide answers without accessing the source data.

In addition to standard operations, the SQL engine can use bitmap indices to efficiently perform special set-based operations using combinations of multiple indices, without accessing source data. For example, to find the total number of 'male survivors', the SQL engine can simply perform a logical AND operator between bitmaps *Survivor* = '*Yes*' and *Gender* = '*Male*', and then count the number of '1' in the result bitmap (Table 4). Hence, 367 men survived the shipwreck. Note that, to obtain the result, the SQL engine does not require to browse the TITANIC table.

6.3 Bitmap indices for building decision trees

In order to build decision trees using bitmap indices, for an initial training set, we create its associated set of bitmap indices for both the predictive attributes and the class attribute. For the root node of the decision tree, the frequency of each class is obtained by simply counting the total number of '1' values in the corresponding bitmap. For each other node in the decision tree, we compute a new set of bitmaps, each one corresponding to a class in the node. The bitmap characterising each class in the current node is obtained by applying the AND operator between the bitmap associated to the node and the bitmaps corresponding to the successive related nodes from the root to the current node. To compute the frequency of each class in this node, we count the total number of '1' in the result bitmap. Since the information gain is based on population frequencies, it is also computed with bitmap indices.

6.4 Running example

To illustrate our method, let us take the TITANIC database presented in Table 1 as an example.

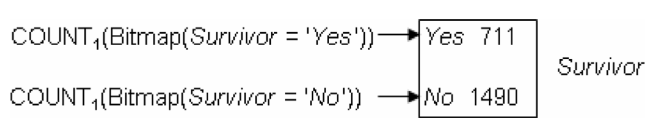
For each predictive attribute and the class attribute, we create its corresponding bitmap index (Table 5). Thus, our new learning population is precisely composed of these four bitmap indices. Hence, we apply the decision tree building method directly on this set of bitmap indices instead of the whole TITANIC database.

Table 5 Bitmap indices for the TITANIC database

	RowId	...	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Class	Crew	...	1	0	0	1	1	0	0	0	0	0	0	0
	1st	...	0	0	0	0	0	1	0	1	0	0	0	1
	2nd	...	0	0	1	0	0	0	1	0	0	1	0	0
	3rd	...	0	1	0	0	0	0	0	0	1	0	1	0
Age	Child	...	0	0	0	0	0	0	0	0	0	1	0	0
	Adult	...	1	1	1	1	1	1	1	1	1	0	1	1
Gender	Female	...	0	0	0	1	0	0	0	1	0	0	0	1
	Male	...	1	1	1	0	1	1	1	0	1	1	1	0
Survivor	No	...	1	1	1	0	0	0	1	0	0	0	0	0
	Yes	...	0	0	0	1	1	1	0	1	1	1	1	1

To build the root node of the decision tree, we just have to determine the frequency of each class. In our running example, the class attribute *Survivor* has two possible values: '*Yes*' or '*No*'. Thus, we have to determine two sub-populations, one for *Survivor* = '*Yes*' and the other for *Survivor* = '*No*' from the bitmap index of the *Survivor* attribute (Table 3). The frequency of each class in the *Survivor* attribute is obtained by counting the number of '1' in the bitmap associated to *Survivor* = '*Yes*' and in the bitmap associated to *Survivor* = '*No*', respectively (Figure 10).

Figure 10 Root node



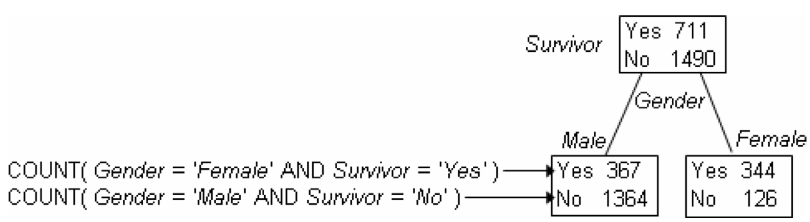
The variation of entropy indicates that the splitting attribute is *Gender*. This attribute has two possible values '*Female*' and '*Male*'. The population of the current node is then divided into two sub-nodes corresponding to the rules *Gender* = '*Male*' and *Gender* = '*Female*', respectively. Each sub-node is composed of two sub-populations that survived or not. To obtain the sizes of these sub-populations, we apply the logical operator AND firstly between the *Gender* = ‘*Male*’ and the *Survivor* = ‘*Yes*’ bitmaps and secondly between the *Gender* = ‘*Male*’ and the *Survivor* = ‘*No*’ bitmaps, as shown in Table 6.

Table 6 *AND_bitmaps* for the node *Gender* = ‘*Male*’

RowId	···	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivor = ‘*Yes*’	...	0	0	0	1	0	1	0	1	1	1	1	1
Gender = ‘*Male*’	...	1	1	1	0	0	1	1	0	1	1	1	0
AND	...	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**
Survivor = ‘*No*’	...	1	1	1	0	1	0	1	0	0	0	0	0
Gender = ‘*Male*’	...	1	1	1	0	0	1	1	0	1	1	1	0
AND	...	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**

To obtain the frequency of the sub-population associated to the rule “*Survivor* = ‘*Yes*’ AND *Gender* = ‘*Male*’ ” (respectively “*Survivor* = ‘*No*’ AND *Gender* = ‘*Male*’ ”), we simply count the total number of ‘1’ in the corresponding *AND_bitmap* (Table 6). The same process is applied for the node corresponding to the rule *Gender* = ‘*Female*’ (Figure 11).

Figure 11 Splitting with the *Gender* attribute



The variation of entropy now indicates that the next splitting attribute is *Class*. From the node *Gender* = ‘*Male*’, we obtain four sub-nodes since the *Class* attribute has four values (‘*1st*’, ‘*2nd*’, ‘*3rd*’ and ‘*Crew*’). For example, to obtain the frequencies of the sub-populations corresponding to the node *Class* = ‘*1st*’, we compute two *AND_bitmaps*, namely (*Gender* = ‘*Male*’ AND *Class* = ‘*1st*’ AND *Survivor* = ‘*Yes*’) and (*Gender* = ‘*Male*’ AND *Class* = ‘*1st*’ AND *Survivor* = ‘*No*’) (Table 7). The sub-populations’ frequencies are then obtained by counting the total number of ‘*1*’ in each *AND_bitmap* obtained.

Table 7 *AND_bitmaps* associated to the node corresponding to the rule *Class* = ‘*1st*’

RowId	···	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Survivor = ‘*Yes*’ AND *Gender* = ‘*Male*’	...	0	0	0	0	0	1	0	0	1	1	1	0
Class = ‘*1st*’	...	0	0	0	0	0	1	0	1	0	0	0	1
AND	...	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
Survivor = ‘*No*’ AND *Gender* = ‘*Male*’	...	1	1	1	0	0	0	1	0	0	0	0	0
Class = ‘*1st*’	...	0	0	0	0	0	1	0	1	0	0	0	1
AND	...	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

6.5 Implementation

The implementation of the ID3 method using bitmap indices takes the form of a PL/SQL stored procedure named *Bitmap_ID*3 under Oracle 10 g. This stored procedure allows us to create the necessary bitmap indices for a given training set and then to build the decision tree. Since Oracle uses B-Tree indices by default, we forced it to use bitmap indices. The nodes of the decision tree are built by using an SQL query that is based on an AND operation applied on its own bitmaps and its parent bitmaps. Then, the obtained *AND_bitmaps* are used to count the population frequency of each class in the node with simple COUNT queries. These counts are used to determine the criterion 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. Similarly, to compute the information gain for a predictive attribute, our implementation uses bitmap indices rather than the whole training set.

6.6 Complexity study

Our objective here is to confirm, from a theoretical point of view, the gain induced by considering the set of bitmap indices rather than the initial training set as the learning set (we denote them as bitmap index-based method and classical method, respectively). For this study we place ourselves in the worst case, i.e. the indices are too large to be loaded in main memory.

Let N be the total number of tuples in the training set, K the number of attributes, L the average length, in bits, of each attribute and A the average number of values of each attribute.

First, we evaluate the size of training sets. The size of the initial training set is $N \times L \times K$ bits. For our bitmap index-based method, this initial training set is replaced by the set of bitmap indices. Thus K bitmap indices are created with an average number of A bitmaps for each index. Each bitmap has a size of N bits. In this case, the size of the training set is $N \times A \times K$ bits. As regards to the size of the training set and thus the loading time, our method is preferable if $A < L$, which corresponds to a majority of cases.

In terms of data reading time, we consider that a bit is read in one time unit. The total number of nodes on the ith depth level can be approximated by A^{i-1}. Indeed we suppose that the obtained decision tree is complete and balanced. To reach level $i + 1$ from an unspecified level i of the tree, each training set must be read as many times as there are predictive attributes remaining at this level, i.e. $(K - i)$.

In the classical method, as the size of the training set is $N \times L \times K$, the reading time for level i (in time units) is $(K - i) \times N \times L \times K \times A^{i-1}$. Hence, to build the whole decision tree in the classical method, the reading time is : $\sum_{i=1}^{K} (K-i) \times N \times L \times K \times A^{i-1}$.

In our bitmap index-based method, the index size is approximated by $N \times A$ bits. To reach level $i + 1$ from an unspecified level i of the tree for a given predictive attribute, the number of index to read is $i + 1$. Thus, at level i, the reading time is: $(i + 1)(K - i)N \times A^i$. Hence, to build the whole decision tree with our bitmap index-based method, the reading time is: $\sum_{i=1}^{K} (i+1)(K-i)N \times A^i$.

To evaluate the gain in time, we build the following ratio:

$$R = \frac{\text{time with classical method}}{\text{time with bitmap index-based method}} = \frac{\frac{KL}{A}\sum_{i=1}^{K}(K-i)\times A^i}{\sum_{i=1}^{K}(K-i)(i+1)\times A^i}.$$

After computing we obtain: $R = \dfrac{\frac{KL}{A}\sum_{i=1}^{K}(K-i)\times A^i}{\sum_{i=1}^{K}(K-i)\times A^i + \sum_{i=1}^{K}i(K-i)\times A^i}$

$$R^{-1} = \frac{A}{KL}\left(1+\frac{\sum_{i=1}^{K}i(K-i)\times A^i}{\sum_{i=1}^{K}(K-i)\times A^i}\right) = \frac{A}{KL}(1+G)$$

As we consider the polynomials of higher degree, G is of complexity K. Thus R^{-1} is of complexity A/L. Indeed $R^{-1} = (A/KL)(1 + K) = (A/L)[1+(1/K)]$ and $1/K$ is insignificant. Our method is interesting if the ratio R^{-1} is lower than one, that means if $A < L$, which corresponds to a majority of cases.

7 Performance

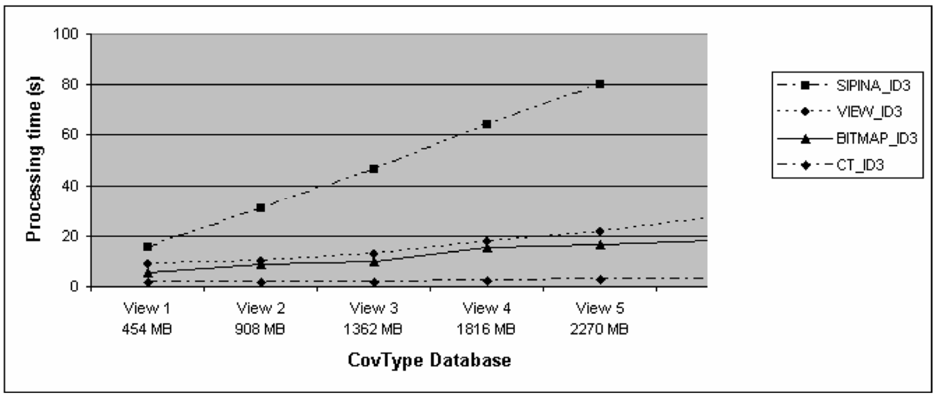
In order to validate our integrated implementation of data mining methods and to compare its performance with an in-memory implementation, we carried out tests on different views from the CovType database[1]. The CovType database contains 581,012 tuples defined by 54 predictive attributes and one class (with seven distinct values). We created five views, each one containing a part of the CovType database and defined by three predictive attributes (each one having five values) and the class. The predictive attributes we used and the size of each view are provided in Figure 12. These tests have been performed on a PC computer with 1.50 GHz and 512 MB of RAM running the Personal Oracle DBMS version 10 g.

Figure 12 Views used in CovType tests

View name	Predictive attributes used	View size (in tuples)	View size (in MB)
view1	1,2,3	116202	454
view2	4,5,6	232404	908
view3	7,8,9	348607	1362
view4	1,4,10	464810	1816
view5	2,5,8	581012	2270

Figure 13 shows the results achieved with our different implementations of ID3. The classical in-memory method using the Sipina software (Zighed and Rakotomalala, 1996), the view-based, the contingency table-based and the bitmap index-based implementations are, respectively, labelled as *Sipina_ID*3, *View_ID*3, *CT_ID*3 and *Bitmap_ID*3. For integrated approaches, we add the time required for building bitmap indices and the contingency table to processing time. In opposition, processing time with *Sipina_ID*3 includes loading time, since it is necessary to load the data from the database into the memory each time the algorithm is executed.

Figure 13 Performance comparison of ID3 implementations



First of all, we note that for databases larger than 2,270 MB, with the hardware configuration used for the tests, Sipina is unable to build the decision tree, whereas our integrated methods can. Sipina is indeed limited by the size of the memory.

Moreover, our results clearly underline the gain induced by our integrated approach, compared with the classical in-memory, approach. The processing time for *Sipina_ID*3, indeed, increases from about 16 to 80 sec when the view size is multiplied by 5. Thus, the processing time for *Sipina_ID*3 is multiplied by about 5, whereas it is multiplied by 3 for view-based and bitmap index-based methods, and by a little more than 1 for the contingency table-based method.

Now, if we compare our different integrated methods, processing time increases more smoothly. The processing time increase is almost identical for *View_ID*3, and *Bitmap_ID*3 (from about 9 to 22 sec for *View_ID*3, and from about 5 to 16 sec for *Bitmap_ID*3). Processing time for *CT_ID*3 is almost constant (from about 2 to 3 sec).

Our experimental results also demonstrate that the contingency table-based method is the best integrated method. For *CT_ID*3, the induced gain mainly depends on the size of the contingency table, which is generally considerably smaller than the size of the initial training set. Nevertheless, in extreme cases, the size of the contingency table may be so close to that of the whole training set so 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.

*View_ID*3 is the slowest integrated method. In this case, processing times remain quite long because of multiple accesses to the database, because it does not use any optimisation tool. The bitmap index-based method is about 30% faster than *View_ID*3 on an average. This result was expected since using bitmap indices avoids many data accesses.

Finally, we can say that our integrated methods are particularly interesting for large databases. Sipina is indeed very fast for computing and very slow for loading data, whereas our integrated methods bear the opposite behaviour; and loading time increases quicker than computing time when the database grows larger. The use of a contingency table as an optimisation tool improves processing times the most significantly.

8 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 exploits separate data mining pieces of software that operate in main memory. The main objective in this approach is then to reduce the size of databases, either by using techniques for pre-processing data or by sampling. The integrated approach comprises processing of the 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.

Following the integrated approach, we proposed in this paper a framework to offer to DBMS users the online data mining operators for mining large databases without size limit and in acceptable processing times. We proposed three integrated methods (a view-based method, a contingency table-based method and a bitmap index-based method) for applying decision tree algorithms on large databases. Each method is based on a specific database tool, namely relational views, contingency table and bitmap indices, respectively.

To validate our online data mining approach, we have implemented three decision tree building methods (ID3, C4.5, CART)[2] under Oracle 10g, as a PL/SQL package named *decision_tree* that is available online[3].

Moreover, we carried out tests on different data sets to compare our different integrated methods with the classical in-memory method. Our experimentation clearly underlined the efficiency of our integrated methods when the database is large. We showed that we could process very large databases without any size limit, while Sipina could not. In addition, we showed that our contingency table-based method presented the best processing time. This result could be expected since it is based on aggregated data that reduce the size of the initial training set. Note that in-memory data mining methods could also use contingency tables instead of original learning sets to improve their processing time.

The perspectives opened by this study are numerous. First, we plan to add in the *decision_tree* package other procedures to supplement the offered data mining tools, such as sampling, missing values management, learning validation techniques and non-supervised learning methods.

We also aim to adapt our integrated approach to mine data warehouses, since they can be stored as relational databases. For example, our contingency-table based method can be performed on relational data cubes by applying the SUM function.

Finally, most of data mining research has concentrated on the single table case. We are currently extending our integrated approach to deal with multiple relational tables. Our first idea comprises using bitmap join indices. Then, we can talk about online database mining, which incorporates the ability to directly access the data stored in a database (several related tables) rather than online data mining (one single table).

Acknowledgement

The authors are very grateful to the anonymous reviewers and the editor of this article, whose comments and suggestions greatly helped to improve this work.

References

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. and Verkamo, A.I. (1996) 'Fast discovery of association rules', In *Advances in Knowledge Discovery and Data Mining* (pp.307–328). Boston, MA: AAAI/MIT Press.

Bentayeb, F. and Darmont, J. (2002) 'Decision tree modeling with relational views', Paper presented at the *XIIIth International Symposium on Methodologies for Intelligent Systems (ISMIS 02)*, Lyon, France, Vol. 2366 of *LNAI*, pp.423–431. Springer. In proceedings.

Bentayeb, F., Darmont, J. and Udréa, C. (2004) 'Efficient integration of data mining techniques in DBMSs', Paper presented at the *VIIIth International Database Engineering and Applications Symposium (IDEAS 04)*, Coimbra, Portugal, (pp.59–67). IEEE Computer Society. In proceedings.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984) *Classification and Regression Trees*. UK: Wadsworth.

Calders, T., Goethals, B. and Prado, A. (2006) 'Integrating pattern mining in relational databases', Paper presented at the *Xth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 06)*, Berlin, Germany. LNAI, Springer. In proceedings.

Chauchat, J.H. and Rakotomalala, R. (2001) 'Sampling strategy for building decision trees from very large databases comprising many continuous attributes', In *Instance Selection and Construction for Data Mining*, Vol. 608 of *International Series in Engineering and Computer Science*, pp.171–188. New York: Kluwer Academic Publishers.

Chaudhuri, S. (1998) 'Data mining and database systems: where is the intersection?', *Data Engineering Bulletin*, Vol. 21, pp.4–8.

Codd, E.F. (1993) 'Providing OLAP (On-Line Analytical Processing) to user-analysts: an IT mandate', Techinical report E.F. Codd and Associates.

Dunkel, B. and Soparkar, N. (1999) 'Data organization and access for efficient data mining', Paper presented at the *XVth International Conference on Data Engineering (ICDE 99)*, Sydney, Australia, pp.522–529. IEEE Computer Society. In proceedings.

Elfeky, M.G., Saad, A.A. and Fouad, S.A. (2000) 'QDMQL object data mining query language', Paper presented at the *International Symposium on Objects and Databases (ECOOP 00)*, Sophia Antipolis, France, Vol. 1944 of *LNCS*, pp.128–140. Springer. In proceedings.

Favre, C. and Bentayeb, F. (2005) 'Bitmap index-based decision trees', Paper presented at the *XVth International Symposium on Methodologies for Intelligent Systems (ISMIS 05)*, Saratoga Springs, New York, USA, Vol. 3488 of *LNCS*, pp.65–73. Springer. In proceedings.

Feng, L. and Dillon, T.S. (2005) 'An XML-enabled data mining query language: XML-DMQL', *Int. J. Business Intelligence and Data Mining*, Vol. 1, pp.22–41.

Fu, X. and Wang, L. (2005) 'Data dimensionality reduction with application to improving classification performance and explaining concepts of data sets', *Int. J. Business Intelligence and Data Mining*, Vol. l, pp.65–87.

Gehrke, J., Ramakrishnan, R. and Ganti, V. (2000) 'RainForest – a framework for fast decision tree construction of large datasets', *Data Mining and Knowledge Discovery*, Vol. 4, pp.127–162.

Geist, I. and Sattler, K.U. (2002) 'Towards data mining operators in database systems: algebra and implementation', Paper presented at the *IInd International Workshop on Databases, Documents, and Information Fusion (DBFusion 02) – Information Integration and Mining in Databases and on the Web,* Karlsruhe, Germany. In proceedings.

Han, J., Fu, Y., Wang, W., Koperski, K. and Zaiane, O. (1996) 'DMQL: a data mining query language for relational databases', Paper presented at the *SIGMOD 96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 96)*, Montreal, Canada, pp.27–34. In proceedings.

Imielinski, T. and Virmani, A. (1999) 'MSQL: a query language for database mining', *Data Mining and Knowledge Discovery*, Vol. 3, pp.373–408.

Lee, H.J., Park, W-H. and Park, D-S. (2003) 'An efficient algorithm for mining quantitative association rules to raise reliance of data in large databases', Paper presented at the *IIIrd International Conference on Hybrid Intelligent Systems (HIS 03)*, Melbourne, Australia, Vol. 105 of *Frontiers in Artificial Intelligence and Applications*, pp.672–681. IOS Press. In proceedings.

Liu, H. and Motoda, H. (1998) *Feature Selection for Knowledge Discovery and Data Mining*. New York: Kluwer Academic Publishers.

Lu, H. and Liu, H. (2000) 'Decision tables: scalable classification exploring RDBMS capabilities', Paper presented at the *XXVIth International Conference on Very Large Data Bases (VLDB 00)*, Cairo, Egypt, pp.373–384. Morgan Kaufmann. In proceedings.

Luo, C., Thakkar, H., Wang, H., and Zaniolo, C. (2005) 'A native extension of SQL for mining data streams', Paper presented at the *XXIVth ACM SIGMOD International Conference on Management of Data (SIGMOD 05)*, Baltimore, Maryland, USA, pp.873–875. ACM Press. In proceedings.

Meo, R. (2003) 'Optimization of a language for data mining', Paper presented at the *ACM Symposium on Applied computing (SAC 03)*, Melbourne, Florida, USA, pp.437–444. ACM Press. In proceedings.

Meo, R., Psaila, G. and Ceri, S. (1998) 'An extension to SQL for mining association rules', *Data Mining and Knowledge Discovery*, Vol. 2, pp.195–224.

Morzy, T. and Zakrzewicz, M. (1998) 'Group bitmap index: a structure for association rules retrieval', Paper presented at the *IVth International Conference on Knowledge Discovery and Data Mining (KDD 98)*, pp.284–288. AAAI Press. In proceedings.

O'Neil, P.E. (1987) 'Model 204 architecture and performance', Paper presented at the *IInd International Workshop on High Performance Transaction Systems*, Asilomar, California, USA, Vol. 359 of *LNCS*, pp.40–59. Springer-Verlag. In proceedings.

O'Neil, P.E. and Quass, D. (1997) 'Improved query performance with variant indexes', Paper presented at the *ACM SIGMOD International Conference on Management of Data (SIGMOD 97)*, Tucson, Arizona, USA, pp.38–49. ACM Press. In proceedings.

Oracle (2001) 'Oracle 9i data mining', White paper.

Ordonez, C. (2006) 'Integrating K-means clustering with a relational DBMS using SQL', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, pp.188–201.

Quinlan, J.R. (1986) 'Induction of decision trees', *Machine Learning*, Vol. 1, pp.81–106.

Quinlan, J.R. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ramesh, G., Maniatty, W. and Zaki, M.J. (2002) 'Indexing and data access methods for database mining', Paper presented at the *VIIth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 02)*, Madison, Wisconsin, USA. In proceedings.

Sarawagi, S., Thomas, S. and Agrawal, R. (1998) 'Integrating mining with relational database systems: alternatives and implications', Paper presented at the *ACM SIGMOD International Conference on Management of Data (SIGMOD 98)*, Seattle, Washington, USA, pp.343–354. ACM Press. In proceedings.

Scheffer, T. and Wrobel, S. (2002) 'A scalable constant-memory sampling algorithm for pattern discovery in large databases', Paper presented at the *VIth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 02)*, Helsinki, Finland, Vol. 2431 of *LNCS*, pp.397–409. Springer. In proceedings.

Soni, S., Tang, Z. and Yang, J. (2001) 'Performance study of Microsoft data mining algorithms', Technical report Microsoft Corp.

Toivonen, H. (1996) 'Sampling large databases for association rules', Paper presented at the *XXIInd International Conference on Very Large Data Bases (VLDB 96)*, Bombay, India, pp.134–145. Morgan Kaufmann. In proceedings.

Wang, H., Zaniolo, C. and Luo, C.R. (2003) 'ATLAS: a small but complete SQL extension for data mining and data streams', Paper presented at the *XXIVth International Conference on Very Large Data Bases (VLDB 03)*, Berlin, Germany, pp.1113–1116. Morgan Kaufmann.

Zighed, D.A. and Rakotomalala, R. (1996) 'SIPINA-W(c) for Windows: user's guide', Technical report, ERIC laboratory University of Lyon 2, France.

Notes

[1]http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/

[2]In this paper, we have detailed the algorithm and results only for the ID3 method.

[3]http://bdd.univ-lyon2.fr/download/decision_tree.zip