Efficient Integration of Data Mining Techniques in Database Management Systems
Fadila Bentayeb, Jérôme Darmont, Cédric Udréa

To cite this version:

HAL Id: hal-00321982
https://hal.archives-ouvertes.fr/hal-00321982
Submitted on 16 Sep 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 \( p(\text{Class } #i) \) 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 \text{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 \text{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 \text{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 \text{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 \( T S \) 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
Figure 1. Example of decision tree

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:

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

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:

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

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

$$G(S_K) = h_s(s_j) - E(S_K)$$

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.
CREATE VIEW CT_view AS
SELECT A_1, ..., A_n, C, COUNT(*) AS Frequency
FROM T
GROUP BY A_1, ..., A_n, C

Figure 3. Performance comparison of View_ID3 and Sipina_ID3

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:

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

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:

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

(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
### Table 1. TITANIC training set sample

<table>
<thead>
<tr>
<th>Class</th>
<th>Age</th>
<th>Gender</th>
<th>Survivor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Adult</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>2nd</td>
<td>Adult</td>
<td>Male</td>
<td>No</td>
</tr>
<tr>
<td>Crew</td>
<td>Adult</td>
<td>Female</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd</td>
<td>Child</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>1st</td>
<td>Adult</td>
<td>Female</td>
<td>No</td>
</tr>
<tr>
<td>1st</td>
<td>Child</td>
<td>Female</td>
<td>Yes</td>
</tr>
<tr>
<td>Crew</td>
<td>Adult</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>2nd</td>
<td>Child</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd</td>
<td>Adult</td>
<td>Female</td>
<td>No</td>
</tr>
<tr>
<td>1st</td>
<td>Child</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>2nd</td>
<td>Adult</td>
<td>Female</td>
<td>No</td>
</tr>
<tr>
<td>2nd</td>
<td>Adult</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd</td>
<td>Adult</td>
<td>Male</td>
<td>No</td>
</tr>
<tr>
<td>Crew</td>
<td>Adult</td>
<td>Male</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd</td>
<td>Adult</td>
<td>Male</td>
<td>No</td>
</tr>
<tr>
<td>2nd</td>
<td>Adult</td>
<td>Female</td>
<td>Yes</td>
</tr>
<tr>
<td>Crew</td>
<td>Adult</td>
<td>Male</td>
<td>No</td>
</tr>
<tr>
<td>Crew</td>
<td>Adult</td>
<td>Female</td>
<td>No</td>
</tr>
<tr>
<td>1st</td>
<td>Child</td>
<td>Female</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd</td>
<td>Adult</td>
<td>Male</td>
<td>No</td>
</tr>
</tbody>
</table>

---

**Figure 5. Relational view associated to the TITANIC contingency table**

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

### 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 \textit{View\_ID3} and the contingency table-based approach \textit{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 DBMS client and server were on the same machine, hence no network traffic can interfere with our per-

Table 2. Classical contingency table for TITANIC

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

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 presents 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-


Table 4. Views used in CovType tests

<table>
<thead>
<tr>
<th>View name</th>
<th>Predictive attributes used</th>
<th>View size</th>
<th>Size of contingency table</th>
</tr>
</thead>
<tbody>
<tr>
<td>view1</td>
<td>1,2,3</td>
<td>116202 tuples</td>
<td>322 tuples</td>
</tr>
<tr>
<td>view2</td>
<td>4,5,6</td>
<td>232404 tuples</td>
<td>265 tuples</td>
</tr>
<tr>
<td>view3</td>
<td>7,8,9</td>
<td>348607 tuples</td>
<td>214 tuples</td>
</tr>
<tr>
<td>view4</td>
<td>1,4,10</td>
<td>464810 tuples</td>
<td>292 tuples</td>
</tr>
<tr>
<td>view5</td>
<td>2,5,8</td>
<td>581012 tuples</td>
<td>264 tuples</td>
</tr>
</tbody>
</table>

Figure 6. Performance comparison of CT_ID3 and View_ID3

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.
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 topped. 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_free 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_free 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


