Efficient integration of data mining techniques in DBMSs

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

We propose in this paper a new approach for applying data mining algorithms, 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 in this paper 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: Integration, Databases, Data mining, Decision trees, Contingency table, Supervised machine learning, Performance.
1 Introduction

The application of data mining operators to huge databases is an interesting challenge. However, data mining algorithms usually operate only on main memory data structures, and more precisely on attribute-value tables. This limits the size of the processed databases. Traditional data mining methods thus use techniques for preprocessing the data, such as feature selection [8] or sampling [3].

In order to apply data mining algorithms to large databases, several approaches have been proposed these last years. They consist in integrating data mining methods in Database Management Systems (DBMSs) [4]. The first studies about integrating data analysis methods into DBMSs came with the development of data warehousing, and on-line analysis (OLAP) in particular [5]. Other related research efforts include the generation of association rules and their generalization [9, 13]. However, few studies exist regarding the integration of classical data mining or analysis techniques such as clustering or classification. In this domain, research indeed principally focus on the application of data mining methods on large databases [1, 6]. Nevertheless, most DBMS vendors included data mining methods into their products [7, 10, 14]. 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).

This is why we proposed a different approach for integrating decision tree-like data mining methods, using only the tools offered by DBMSs [2]. More
precisely, we implemented the ID3 method [11] within the Oracle DBMS as a PL/SQL stored procedure, by making use of relational views. In this approach, each node of the decision tree, which represents a subpopulation of its parent node, is represented as a relational view built from its parent view (the root view corresponds to the whole training set). These views are used to number the population strengths of each class within a node, which helps selecting a partitioning criteria. We showed that we could process very large databases with this approach, 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 includes the aggregate population strengths, 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.
This remainder of this paper is organized as follows. Section 2 details our contingency table-based decision tree construction method. Section 3 presents the experimental results and the complexity study that validate our proposal. Finally, Section 4 concludes this paper and presents research perspectives.

2 Contingency table-based decision tree construction

2.1 Construction of the contingency table

Since we apply data mining algorithms within DBMSs, the contingency table corresponding to a given training dataset is computed by a simple SQL query. Let $TS$ be a training set defined by $n$ predictive attributes $A_1, \ldots, A_n$ and one class attribute $C$. The associated contingency table $CT$ is obtained by running the SQL query displayed in Figure 1. We term "Population" the additional attribute that represents the aggregate population strength.

\begin{verbatim}
CREATE VIEW CT_view AS
SELECT A_1, \ldots, A_n, C, COUNT(*) AS Population
FROM TS
GROUP BY A_1, \ldots, A_n, C
\end{verbatim}

Figure 1: Relational view associated to contingency table $CT$

Since the contingency table contains aggregate population strengths, its size is usually smaller than the whole training set (and at most as large as the training set). Therefore, the gain in terms of processing time is normally significant.
2.2 Running example

To illustrate how our approach works, we use the TITANIC training set, which contains 2201 objects (or tuples, in the relational databases vocabulary) defined by three predictive attributes: Class (1st, 2nd, 3rd, Crew), Age (Adult, Child), and Gender (Male, Female); and one class attribute: Survivor (Yes, No). The SQL query associated to this dataset’s contingency table is provided in Figure 2. Its result contains only 24 tuples.

![SQL query]

Figure 2: Relational view associated to the TITANIC contingency table

2.3 Entropy and information gain

Decision trees may be viewed as a succession of smaller and smaller partitions of an initial training set. Decision tree construction methods apply successive criteria on the training population to obtain these partitions, wherein the size of one class is maximized. In the ID3 algorithm [11], 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:

$$h_s(s_k) = - \sum_{i=1}^{c} \frac{n_{ik}}{n_k} \log_2 \frac{n_{ik}}{n_k}$$ (1)
where $n_k$ is the population strength 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:

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

(2)

where $n_j$ is the population strength 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)

### 2.4 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 adaptations 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 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 strength of a subpopulation satisfying a given set of rules $E_r$ (e.g., $Age = Child \ AND \ Gender = Female$) by summing the values of the Population 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 of a predictive attribute. Indeed, as presented in section 2.3, the usual calculation of the information gain for an attribute having \( k \) possible values and with a class attribute having \( c \) possible values is:

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

(4)

where \( n_j \) is the node population strength, \( n_k \) is the population strength of the subnode having value \( V_k \) for the predictive attribute, \( n_{ik} \) is the population strength 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:

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

(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 population strengths \( (n_j, n_k, n_{ik}) \) and we obtain \( n_k \) by summing the \( n_{ik} \) and \( n_j \) by summing the \( n_k \).
3 Validation

3.1 Implementation

We used 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 \textit{decision\_tree} that is available on-line\(^1\). We have tested this implementation under Oracle version 8i and 9i.

3.2 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 the TITANIC training set presented in section 2.2.

We first compared the processing times of both our implementations of ID3: the view-based approach, named \textit{View\_ID3}, and the contingency table-based approach \( CT_{ID3} \). These tests have been carried out in the same environment: a PC computer with 128 MB of RAM and the Personal Oracle DBMS version 9i. The use of Personal Oracle ensured that the DMBS client and server were on the same machine, hence no network traffic could interfere with our performance measurements. The results we obtained clearly underlined the gain induced by \( CT_{ID3} \), that had a processing time 24 times smaller than \textit{View\_ID3} on an average, while producing the same result.

Then, we compared the processing times of \( CT_{ID3} \) to those of a clas-

\(^1\)http://bdd.univ-lyon2.fr/download/decision\_tree.zip
sical, in-memory implementation of ID3 available in the Sipina data mining software [12]. We observed that our contingency table-based approach presented similar processing times than *Sipina_ID3*, while allowing to process bigger databases (Figure 3).

![Figure 3: Performance comparison of CT_ID3 and Sipina_ID3](image)

In conclusion, we hint the effectiveness of our approach, since generally, it 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 considerably. 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.
3.3 Complexity study

Our objective is to compare the complexity of our integrated approaches View_ID3 and CT_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 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 population strengths at this level corresponds to the population strength 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, whose tuples bear \( K \) predictive attributes, one class attribute, and a population strength attribute, for total size of \( K + 2 \) attributes and thus a writing time of \( T(K + 2) \). To compute the contingency table, we read the whole database once. The reading time is thus \( N(K + 1) \). When reaching level \( i + 1 \) from level \( i \), we read all the \( T \) tuples of size \( K + 2 \) attributes \( (K - i) \) times, for a total time by level of \( (K - i)T(K + 2) \).

In our contingency table-based approach, we first create the contingency table, thus a writing time of \( T \). 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 \( CTJD3 \), 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.
4 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 the decision tree (View ID3) [2]. 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. 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. Other tests on other databases are also currently in process. These tests aim at empirically determining the gain in processing time when the number of attributes and/or the number of tuples increase. 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 techniques.

References


