Intégration efficace de méthodes de fouille de données dans les SGBD

Cédric Udréa, Fadila Bentayeb
Jérôme Darmont, Omar Boussaid

ERIC – Université Lumière Lyon 2
5 avenue Pierre Mendès-France – 69676 Bron Cedex – France
{cudrea | bentayeb | jdarmont | boussaid}@eric.univ-lyon2.fr

Résumé. Cet article présente une nouvelle approche permettant d'appliquer des algorithmes de fouille, en particulier d'apprentissage supervisé, à de grandes bases de données et en des temps de traitement acceptables. Cet objectif est atteint en intégrant ces algorithmes dans un SGBD. Ainsi, nous ne sommes limités que par la taille du disque et plus par celle de la mémoire. Cependant, les entrées-sorties nécessaires pour accéder à la base engendrent des temps de traitement longs. Nous proposons donc dans cet article une méthode originale pour réduire la taille de la base d'apprentissage en construisant sa table de contingence. Les algorithmes d'apprentissage sont alors adaptés pour s'appliquer à la table de contingence. Afin de valider notre approche, nous avons implémenté la méthode de construction d'arbre de décision ID3 et montré que l'utilisation de la table de contingence permet d'obtenir des temps de traitements équivalents à ceux des logiciels classiques.

Mots Clés : Intégration, Bases de données, Fouille de données, Arbres de décision, Vues relationnelles, Table de contingence, Apprentissage supervisé, Performance.

1 Introduction

L'application d'opérateurs de fouille de données sur de grandes bases de données est un enjeu intéressant. Cependant, les algorithmes de fouille de données ne peuvent opérer que sur des structures en mémoire de type tableau attributs-valeurs, ce qui limite la taille des bases à traiter. De ce fait, les méthodes classiques de fouille de données utilisent des méthodes de pré-traitement sur les données, telles que la sélection de variables [Lia et Motoda, 1998] ou l'échantillonnage [Chauchat, 2002].

Afin d'appliquer des algorithmes de fouille de données sur de grandes bases de données, de nouvelles voies de recherche sont apparues ces dernières années. Elles consistent à intégrer des méthodes de fouille dans les Systèmes de Gestion de Bases de Données (SGBD) [Chaudhuri, 1998]. L'une des premières avancées dans le domaine de l'intégration de méthodes d'analyse des données dans les SGBD a été amorcée par l'avènement des entrepôts de données et de l'analyse en ligne (OLAP) en particulier [Codd, 1993]. D'autres travaux de recherche ont concerné l'intégration des méthodes de règles d'association et de leur généralisation [Meo *et al.*, 1996] [Sarawagi *et al.*, 1998]. En revanche, il existe peu de travaux d'intégration dans les SGBD de méthodes classiques de fouille

ou d'analyse de données telles que le groupement ou la classification. En effet, dans ce domaine, la plupart des travaux concernent plutôt l'application de méthodes de fouille sur de grandes bases de données [Agrawal *et al.*, 1996] [Gehrke *et al.*, 1998]. Néanmoins, la plupart des éditeurs de logiciels ont doté leurs SGBD de méthodes de fouille de données [Microsoft, 2000] [IBM, 2001] [Oracle, 2001] [Soni *et al.*, 2001]. Cependant, cette intégration se présente sous forme de boîtes noires réalisées grâce à des extensions de SQL ou nécessitant l'utilisation d'interfaces de programmations (API). C'est pourquoi nous avons proposé [Bentayeb et Darmont, 2002] une approche d'intégration de méthodes de fouille de données de type "arbre de décision" en n'utilisant que les outils offerts par les SGBD. Plus précisément, nous avons implémenté la méthode ID3 dans le SGBD Oracle sous forme d'une procédure stockée PL/SQL en exploitant les vues relationnelles. Avec cette approche, nous avons montré qu'il est possible de traiter de grandes bases de données sans limitation de taille. Cependant les temps de traitement demeurent longs en raison des accès multiples à la base.

Afin d'améliorer ces temps de traitements, la préparation des données au processus de fouille devient cruciale. Nous proposons dans cet article une méthode originale pour atteindre cet objectif. Il s'agit de construire une table de contingence, c'est-à-dire une table des populations agrégées, de taille beaucoup plus réduite que la base d'apprentissage de départ. Nous avons ensuite adapté les méthodes de fouille de données pour qu'elles puissent s'appliquer à la table de contingence obtenue. En l'état actuel de nos connaissances, aucune méthode de fouille de données n'utilise une telle méthode de préparation de données.

Pour illustrer et valider notre approche, nous avons adapté l'implémentation d'ID3 pour l'appliquer sur la table de contingence obtenue à partir de la base d'apprentissage de départ. Nous montrons que l'utilisation de la table de contingence permet de nettes améliorations, en termes de temps de traitement, par rapport à notre première implémentation d'ID3. De plus, ces temps de traitement sont équivalents à ceux des logiciels classiques.

Cet article est organisé de la façon suivante. La Section 2 présente le principe de notre approche d'intégration des méthodes de fouille de données à base d'arbres de décision, et en particulier l'intégration de la méthode ID3 dans le SGBD Oracle. La Section 3 expose le principe de notre approche utilisant une préparation de données basée sur la construction de la table de contingence. La Section 4 présente les résultats expérimentaux et l'étude de complexité qui valident notre approche. La Section 5 conclut cet article et expose nos perspectives de recherche.

2 Intégration de méthodes de construction d'arbres de décision dans un SGBD

2.1 Principe

Les arbres de décision sont des outils d'apprentissage qui produisent des règles du type "si-alors" [Zighed et Rakotomalala, 2000]. Ils utilisent en entrée un ensemble d'objets (n-uplets) décrits par des variables (attributs). Chaque objet appartient à une classe, les classes étant mutuellement exclusives. Pour construire un arbre de décision,

il est nécessaire de disposer d'une population d'apprentissage (table ou vue) constituée d'objets dont la classe est connue. Le processus d'apprentissage consiste ensuite à déterminer la classe d'un objet quelconque d'après la valeur de ses variables.

Les méthodes de construction d'arbres de décision segmentent la population d'apprentissage afin d'obtenir des groupes au sein desquels l'effectif d'une classe est maximisé. Cette segmentation est ensuite réappliquée de façon récursive sur les partitions obtenues. La recherche de la meilleure partition lors de la segmentation d'un noeud revient à rechercher la variable la plus discriminante pour les classes. C'est ainsi que l'arbre (ou plus généralement le graphe) est constitué.

Finalement, les règles de décision sont obtenues en suivant les chemins partant de la racine de l'arbre (la population entière) jusqu'à ses feuilles (populations au sein desquelles une classe représente la majorité des objets). La Figure 1 montre un exemple d'arbre de décision ainsi que les règles associées. p(Classe i) représente la probabilité d'un objet d'appartenir à la Classe numéro i.

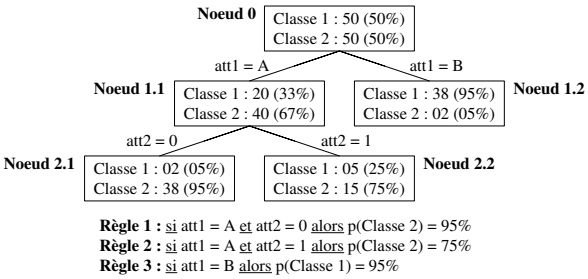

FIG. 1 – Exemple d'arbre de décision

Pour pouvoir appliquer les méthodes de fouille de données sur des bases volumineuses, l'une des nouvelles voies de recherche consiste à intégrer ces méthodes au sein des SGBD, le but étant d'utiliser uniquement les outils offerts par ces derniers. Dans l'approche présentée dans [Bentayeb et Darmont, 2002], la racine d'un arbre de décision est représentée par une vue relationnelle qui correspond à la population d'apprentissage entière. Comme chaque noeud de l'arbre de décision représente une sous-population de son noeud parent, nous associons à chaque noeud une vue construite à partir de sa vue parente. Ces vues sont ensuite utilisées pour dénombrer les effectifs de chaque classe dans le noeud. Ces comptages servent finalement à déterminer le critère de partitionnement des noeuds en sous-partitions ou à conclure qu'un noeud est une feuille. La Figure 2 présente à titre d'exemple les commandes SQL permettant de créer les vues associées à l'arbre de décision de la Figure 1.

2.2 Entropie et gain d'information

Dans l'algorithme ID3 (*Induction Decision Tree*) [Quinlan, 1986], le pouvoir discriminant d'une variable pour la segmentation d'un noeud est exprimé par une variation d'entropie. L'entropie h_s d'un noeud s_i (plus présisément, son entropie de Shannon)

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

FIG. 2 – Vues relationnelles associées à l'arbre de décision exemple

est :

$$h_s(s_i) = -\sum_{j=1}^{c} \frac{n_{ji}}{n_i} \log_2 \frac{n_{ji}}{n_i} \tag{1}$$

où n_i est l'effectif de s_i, n_{ji} le nombre d'objets de s_i qui appartiennent à la classe c_j et c la cardinalité de la classe. L'information portée par une partition S_K de K noeuds est alors la moyenne pondérée des entropies :

$$E(S_K) = \sum_{i=1}^{K} \frac{n_i}{n_k} h_s(s_i) \tag{2}$$

où n_k est l'effectif du noeud s_k qui est segmenté. Finalement, le gain informationnel associé à S_K est :

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

Comme $G(S_K)$ est toujours positif ou nul, le processus de construction d'arbre de décision revient à une heuristique de maximisation de $G(S_K)$ à chaque itération et à la sélection de la variable correspondante pour segmenter un noeud donné. L'algorithme s'arrête lorsque $G(S_K)$ devient inférieur à un seuil (gain minimum) défini par l'utilisateur.

2.3 Discussion

L'avantage d'intégrer des méthodes de fouille de données au sein d'un SGBD est de bénéficier de sa puissance au niveau de l'accès aux données persistantes. En effet, les logiciels classiques nécessitent de charger la base de données en mémoire pour la traiter. Ils sont donc limités au niveau de la quantité de données analysables. Cet état de fait est illustré par la Figure 3, qui représente le temps de construction d'un arbre de décision sur une base dont la taille augmente, avec d'une part des logiciels de fouille classiques (en l'occurrence SIPINA [Zighed et Rakotomalala, 1996] configuré pour utiliser la méthode ID3) et d'autre part, notre implémentation d'ID3 sous Oracle baptisée "Buildtree". Ces tests ont été effectués sur un ordinateur PC disposant de 128 Mo de mémoire vive. Or, dans cette configuration SIPINA ne peut pas traiter des bases dont la taille dépasse 50 Mo.

Ce résultat montre que nos travaux permettent de continuer à traiter des bases de données de grande taille là où les logiciels travaillant en mémoire ne peuvent plus

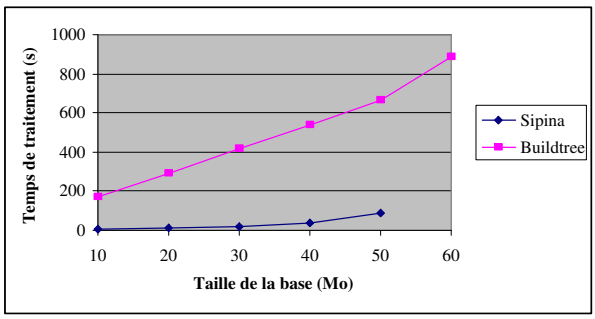

FIG. 3 – Temps de traitement en fonction de la taille de la base

opérer. Cependant si nos résultats en terme de taille de bases à traiter sont prometteurs, les temps de traitements demeurent très longs. Bien que le temps de calcul ne soit pas généralement considéré comme un point critique dans un processus de fouille de données, il est néanmoins nécessaire de le réduire au maximum. Par ailleurs, l'expérience de la Figure 3 met en oeuvre une base de données dont seul le nombre de n-uplets augmente. Or, en schématisant, la complexité des algorithmes de construction d'arbres de décision est linéaire selon le nombre d'objets (n-uplets), mais exponentielle selon le nombre de variables (attributs). Il est donc primordial d'optimiser le temps de traitement des algorithmes de fouille fonctionnant au sein d'un SGBD afin d'obtenir des temps de réponse acceptables.

3 Utilisation d'une table de contingence

3.1 Principe

Pour améliorer les temps de traitement des méthodes de fouille de données intégrées dans un SGBD, nous proposons de réduire la taille de la base d'apprentissage en calculant sa table de contingence. Les algorithmes de fouille de données seront ainsi appliqués sur cette dernière.

Les algorithmes de construction d'arbres de décision procèdent de manière récursive sur chaque noeud pour déterminer la variable prédictive apportant le meilleur gain d'information. Ainsi, pour chaque noeud fils, les mêmes opérations que pour le noeud père sont effectuées mais sur une collection de n-uplets plus restreinte pour chaque fils. Une de ces opérations consiste à déterminer pour chaque valeur de la classe le nombre total de n-uplets pour chacune des valeurs de chaque attribut prédictif. Or, cette façon de procéder entraîne plusieurs lectures de chaque n-uplet, puisque l'ensemble des n-uplets des noeuds fils correspond aux n-uplets de leur père, et que pour chaque fils, on compte les n-uplets de l'ensemble de la collection qui lui est associée. Par conséquent, ces lectures multiples induisent un coût supplémentaire réel alors qu'en définitive, le nombre de n-uplets similaires est plus significatif que les n-uplets eux-mêmes. De plus,

la construction d'un noeud consiste à écrire la sous-population des n-uplets du père correspondant diminuée d'un attribut, donc à écrire de nouveau ces n-uplets.

3.2 Construction de la table de contingence

Dans le contexte des méthodes de fouille intégrées dans un SGBD, la table de contingence est obtenue à l'aide d'une simple requête SQL utilisant un regroupement et une fonction d'agrégat "Count" appliquée sur la table de départ. Un attribut supplémentaire appelé "Population" est donc naturellement obtenu pour représenter l'effectif obtenu pour chaque type de n-uplets dans la table de contingence. De fait, le nombre de n-uplets de la table de contingence est très inférieur ou au pire égal au nombre de n-uplets de la base de départ. Le gain en terme de temps de traitement peut donc être important.

D'une manière générale, pour une base d'apprentissage B définie par les attributs prédictifs $A_1, \ldots, A_n$ et par l'attribut à prédire C, la table de contingence peut être obtenue avec une simple requête SQL (Figure 4).

```
CREATE VIEW Contingence AS SELECT A_1, ..., A_n, C, COUNT(*) AS Population
FROM B GROUP BY A_1, ..., A_n, C
```

FIG. 4 – Vue relationnelle associée à la table de Contingence

3.3 Exemple de TITANIC

Nous illustrons notre approche en utilisant la base d'apprentissage TITANIC qui comporte trois attributs prédictifs Class (1ST, 2ND, 3RD, Crew), Age (Adult, Child) et Gender (Male, Female) ainsi qu'une classe à prédire Survivor (Yes, No) pour une population totale de 2201 individus.

A titre d'exemple, dans [Bentayeb et Darmont, 2002] l'application de la méthode ID3 intégrée dans Oracle ("Buildtree") sur cette base d'apprentissage consiste à créer une succession de vues (partitions) sur la base de départ (2201 individus) qui formeront un arbre de décision. En construisant la table de contingence de TITANIC, la méthode ID3 va s'appliquer sur un nombre d'individus beaucoup plus réduit. En effet, la table de contingence obtenue ne contient que 24 n-uplets (Figure 6).

La requête SQL lui correspondant est présentée dans la Figure 5.

```
CREATE VIEW Contingence AS SELECT Classe, Sexe, Age, Survivant, COUNT(*)
AS Population FROM TITANIC GROUP BY Classe, Sexe, Age, Survivant
```

FIG. 5 – Vue relationnelle associée à la table de Contingence : Exemple du Titanic

Classe	Age	Sexe	Survivant	Population
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

FIG. 6 – Table de contingence de la base TITANIC

3.4 Calcul du gain d'information et de l'entropie

Pour démontrer la pertinence et l'efficacité de notre approche, nous avons implémenté à nouveau la méthode ID3 en tenant compte de la phase de préparation des données. La véritable différence se situe lors du calcul du gain d'information pour chaque attribut prédictif et par conséquent lors du calcul de l'entropie.

Les implémentations de l'algorithme standard sont contraintes pour calculer le gain d'information d'un attribut prédictif de lire tous les n-uplets de la partie de la base de départ correspondant au noeud courant de l'arbre d'induction afin de déterminer la répartition des n-uplets en fonction de chaque valeur de l'attribut prédictif et de chaque valeur de la classe.

Dans notre approche, pour connaître l'effectif de la population d'un noeud obtenu à partir d'un ensemble de critères E_r (Age=Child et Gender=Female, par exemple), il suffit d'effectuer la somme des valeurs de l'attribut "Population" de la table de contingence pour les n-uplets satisfaisant E_r. Cette technique est donc beaucoup plus rapide car elle s'applique sur un nombre bien plus restreint de n-uplets.

Enfin, en modifiant la manière de le calculer, il devient possible de n'effectuer qu'une seule lecture pour connaître le gain d'information d'un attribut prédictif. En effet,

comme nous l'avons vu dans la section 2.2, le calcul normal du gain pour un attribut ayant k valeurs possibles et avec une classe ayant c valeurs possibles est de :

$$gain = (entropie\ du\ noeud) - \sum_{i=1}^{k} \left(\frac{n_i}{n} \times \left(- \sum_{j=1}^{c} \frac{n_{ji}}{n_i} \times log_2 \left(\frac{n_{ji}}{n_i} \right) \right) \right) \quad (4)$$

où n_i est l'effectif du noeud ayant la valeur i pour l'attribut prédictif, n est l'effectif de la population du noeud, n_{ji} est l'effectif du noeud ayant la valeur i pour l'attribut prédictif et la valeur j pour la classe. Or, en développant (4), on obtient :

$$gain = (entropie\ du\ noeud) + \frac{1}{n} \times \sum_{i=1}^{k} \left(n_i \times \frac{1}{n_i} \times \sum_{j=1}^{c} n_{ji} \times log_2 \left(\frac{n_{ji}}{n_i} \right) \right) \quad (5)$$

De plus, $log_2 \frac{a}{b}$ étant égal à $log_2 a - log_2 b$ on obtient d'après (5) :

$$gain = (entropie\ du\ noeud) + \frac{1}{n} \times \sum_{i=1}^{k} \left(\sum_{j=1}^{c} n_{ji} \times (log_2 n_{ji} - log_2 n_i) \right) \quad (6)$$

En développant (6), on obtient :

$$gain = (entropie\ du\ noeud) + \frac{1}{n} \times \left(\sum_{i=1}^{k} \sum_{j=1}^{c} n_{ji} \times log_2 n_{ji} - \sum_{i=1}^{k} \sum_{j=1}^{c} n_{ji} \times log_2 n_i \right) \quad (7)$$

Or

$$\sum_{j=1}^{c} n_{ji} \times log_2 n_i = n_i \times log_2 n_i$$

D'où, d'après (7) :

$$gain = (entropie\ du\ noeud) + \frac{1}{n} \times \left(\sum_{i=1}^{k} \sum_{j=1}^{c} n_{ji} \times log_2 n_{ji} - \sum_{i=1}^{k} n_i \times log_2 n_i \right) \quad (8)$$

En appliquant la Formule 8 sur la table de contingence que nous ne lisons qu'une seule et unique fois, nous obtenons facilement le gain. En effet, dans cette formule il n'est pas nécessaire de connaître au même moment les différents effectifs (n, n_i, n_{ji}) et on obtient n_i par somme sur les n_{ji} et n par somme sur les n_i.

3.5 Implémentation

L'implémentation de la méthode ID3 avec table de contingence est effectuée en PL/SQL, compatible sous Oracle 8i et Oracle 9i, sous la forme d'une procédure stockée nommée "TC_ID3" au sein d'un package de procédures nommé "$decision\_tree$"[1].

L'adaptation de l'algorithme d'ID3 (pour le calcul de l'entropie et du gain d'information) fut donc nécessaire afin qu'il puisse s'appliquer sur la table de contingence.

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

4 Résultats et complexité

4.1 Résultats expérimentaux

Afin de valider notre approche et de comparer ses performances vis-à-vis aussi bien des méthodes classiques que de la méthode intégrée "Buildtree" avec vues relationnelles, nous avons effectué des tests sur la base TITANIC, dont le schéma est donné dans la section 3.3 et qui comporte 2201 n-uplets avec trois attributs prédictifs et un attribut à prédire.

Nous avons comparé les temps de traitements de la méthode ID3 appliquée à la base TITANIC sans préparation de données (BuildTree) et utilisant les vues relationnelles avec la nouvelle version d'ID3 ("TC_ID3") appliquée à la table de contingence. Ces tests sont effectués dans le même environnement (même matériel : un ordinateur PC disposant de 128 Mo de mémoire vive, même version d'Oracle : Oracle 9i, même jeu d'essai : la base TITANIC). Les résultats obtenus montrent l'efficacité de notre approche par rapport à la méthode "Buildtree" puisque le temps de traitement est divisé par 24.

De plus, nous avons comparé les temps de traitements de "TC_ID3" avec SIPINA[2]. Nous pouvons observer que, grâce à l'application d'ID3 sur la table de contingence associée à la base TITANIC, le temps de traitement de notre approche est similaire à celui des méthodes classiques (Figure 7).

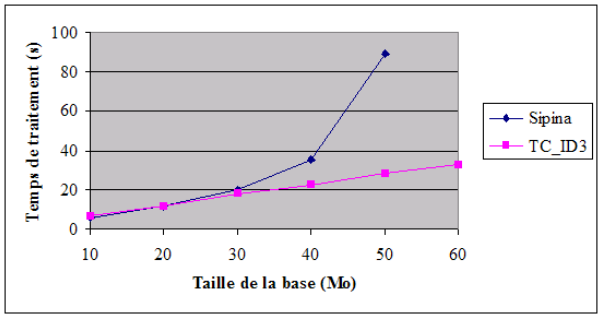

FIG. 7 – Temps de traitement en fonction de la taille de la base

Nous apportons la preuve de l'efficacité de notre approche puisque d'une façon générale, celle-ci réduit considérablement la taille de la base d'apprentissage. Néanmoins, dans des cas extrêmes, il peut arriver que la taille de la table de contingence soit si proche de celle de la base de départ que le gain en devient infime. Ces cas restant toutefois en pratique très rares, l'utilisation de la table de contingence dans le cadre de l'intégration des méthodes de fouille dans les SGBD améliore considérablement les temps de traitement.

[2] http ://eric.univ-lyon2.fr/∼ricco/sipina.html

4.2 Etude de complexité

L'étude de complexité suivante nous permet d'appuyer les résultats expérimentaux obtenus. Soient N le nombre de n-uplets de la base de départ, K le nombre d'attributs prédictifs et T la taille de table de contigence. Notre objectif est de comparer la complexité entre l'algorithme avec vues relationnelles et celui utilisant la table de contingence. Nous considérons que les deux algorithmes sont optimisés dans leur implémentation de telle sorte que seuls les n-uplets nécessaires sont lus.

Dans cette étude, nous nous intéressons au temps passé à la lecture et l'écriture des données car ce sont les opérations les plus coûteuses. Nous considérons qu'un attribut d'un n-uplet est lu ou écrit en une unité de temps. Enfin, nous considérons que l'arbre de décision obtenu est équilibré et complet, c'est-à-dire qu'à chaque niveau de l'arbre, l'union des populations des différents noeuds du niveau équivaut à la base toute entière.

Pour l'algorithme avec vues relationnelles, pour un niveau i quelconque de l'arbre, chaque n-uplet comprend K-i attributs prédictifs plus l'attribut à prédire, soit une taille de (K-i+1). Pour aboutir au niveau i+1, il faut lire chaque noeud autant de fois qu'il existe d'attributs prédictifs à ce niveau, c'est-à-dire (K-i). Comme la somme des populations des noeuds du niveau correspond à la population de la base de départ, il est donc nécessaire de lire N n-uplets d'une taille de (K-i+1) et ce (K-i) fois (nombre de n-uplets $\times$ taille d'un n-uplet $\times$ nombre d'attributs), soit un temps total de lecture pour le niveau i de $N(K-i)(K-i+1) = N(K^2+i^2-2iK+K-i)$. Or, pour obtenir ce niveau, il a fallu écrire les n-uplets correspondants. Les N n-uplets ayant une taille de (K-i+1) chacun, le temps total d'écriture est donc de N(K-i+1).

En rappelant que $\sum_{i=1}^{K} i = K(K+1)/2$ et $\sum_{i=1}^{K} i^2 = K(K+1)(2K+1)/6$ nous obtenons alors une complexité finale de la racine jusqu'aux feuilles (niveau K) égale à :

-en lecture : $N(K^3/3 - K/3)$ unités de temps, donc en NK^3 ;

-en écriture : $NK^2/2 - NK/2$ unités de temps, donc en NK^2.

Pour notre approche, il convient d'abord de créer la table de contingence dont les n-uplets comportent K attributs prédictifs, un attribut à prédire et un attribut population soit une taille de K+2 attributs et donc un temps d'écriture de T(K+2). Pour l'obtenir, il convient de lire intégralement la base de départ une première fois, soit un temps de lecture de N(K+1). A chaque niveau i, pour aboutir au niveau i+1, on lit l'intégralité des T n-uplets de K+2 attributs, et ce (K-i) fois, soit un temps de (K-i)T(K+2) pour chaque niveau.

Comme $\sum_{i=1}^{K}(K-i) = K^2 - K(K+1)/2$, on obtient :
$\sum_{i=1}^{K}(K-i)T(K+2) = T(K+2)(K^2 - K(K+1)/2) = T(K^3+K^2-2K)/2$.

Avec création de la table de contingence, les résultats sont donc :

-en lecture : $T(K^3+K^2-2K)/2 + N(K+1)$ unités de temps,

donc en TK^3 ou en NK si N>TK^2 ;

-en écriture : $T(K+2)$ unités de temps, donc en TK.

Ainsi, en temps de traitement, notre approche apporte une amélioration en N/T ou en K^2 (si N>TK^2) pour la lecture et en NK/T pour l'écriture. Comme N est normalement très supérieur à T, cette amélioration est importante et, de plus, elle augmente avec le nombre d'attributs.

5 Conclusion et Perspectives

L'avantage majeur d'intégrer des méthodes de fouille de données au sein d'un SGBD est de pouvoir traiter de grandes bases de données grâce à sa puissance au niveau de l'accès aux données persistantes. Cependant, le processus de fouille dans les données engendre de nombreuses entrées/sorties qui ralentissent considérablement les temps de traitement. Pour remédier à ce problème, nous avons proposé dans cet article une méthode originale pour la préparation des données. Il s'agit de construire une table de contingence, c'est-à-dire une table des populations agrégées par le biais d'une requête SQL, qui réduit considérablement la taille de la table d'apprentissage de départ. Pour valider notre approche, nous avons implémenté la méthode de construction d'arbre de décision ID3, dont l'algorithme a été adapté à la table de contingence sous forme de procédure stockée PL/SQL nommée "TC_ID3". Ainsi, nous avons montré que l'utilisation de la table de contingence permet de nettes améliorations en termes de temps de traitement par rapport à la fois aux méthodes classiques et à l'approche "Buildtree". De plus, "TC_ID3" présente des résultats similaires en terme de temps de traitement aux méthodes classiques opérant en mémoire.

Afin d'enrichir notre package "*decision_tree*", nous sommes en train d'implémenter d'autres méthodes de fouille de données telles que C4.5 et CART en utilisant la table de contingence. Par ailleurs, nous envisageons de comparer les performances de "TC_ID3" et SIPINA sur de grandes bases de données réelles classiquement employées dans la communauté d'apprentissage. D'autres tests sur d'autres bases sont actuellement en cours. Ces tests ont pour objectif de déterminer empiriquement le gain de temps quand le nombre d'attributs augmente et quand le nombre de n-uplets augmente. De plus, nous prévoyons l'intégration dans le package "*decision_tree*" d'autres procédures pour compléter les outils de fouille de données offerts comme l'échantillonnage, la gestion des valeurs manquantes et la validation.

Références

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

[Bentayeb et Darmont, 2002] F. Bentayeb et J. Darmont. Decision tree modeling with relational views. In *XIIIth International Symposium on Methodologies for Intelligent Systems (ISMIS 2002), Lyon, France*, volume 2366 of *LNAI*, pages 423–431, Heidelberg, Germany, June 2002. Springer Verlag.

[Chauchat, 2002] J. H. Chauchat. *Echantillonnage, validation et généralisation en extraction des connaissances à partir des données.* Mémoire d'habilitation à diriger des recherches, Université Lumière Lyon 2 - France, Janvier 2002.

[Chaudhuri, 1998] S. Chaudhuri. Data mining and database systems : Where is the intersection ? *Data Engineering Bulletin*, 21(1) :4–8, 1998.

[Codd, 1993] E. F. Codd. Providing olap (on-line analytical processing) to user-analysts : An it mandate. Technical report, E.F. Codd and Associates, 1993.

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

[IBM, 2001] IBM. Db2 intelligent miner scoring. http ://www-4.ibm.com/software/data/iminer/scoring, 2001.

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

[Meo *et al.*, 1996] R. Meo, G. Psaila, et S. Ceri. A new sql-like operator for mining association rules. In *22th International Conference on Very Large Data Bases (VLDB 96), Mumbai, India*, pages 122–133. Morgan Kaufmann, 1996.

[Microsoft, 2000] Microsoft. Introduction to ole-db for data mining. http ://www.microsoft.com/data/oledb, July 2000.

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

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1 :81–106, 1986.

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

[Soni *et al.*, 2001] S. Soni, Z. Tang, et J. Yang. Performance study microsoft data mining algorithms. Technical report, Microsoft Corp., 2001.

[Zighed et Rakotomalala, 1996] D. A. Zighed et R. Rakotomalala. Sipina-w(c) for windows : User's guide. Technical report, ERIC laboratory, University of Lyon 2, France, 1996.

[Zighed et Rakotomalala, 2000] D. A. Zighed et R. Rakotomalala. *Graphes d'induction. Apprentissage et Data Mining*. 2000.

Summary

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 DBMS. We are thus only limited by disk capacity, not by available main memory. However, the I/Os that are necessary to access 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 software.

Keywords : Integration, Databases, Data mining, Decision trees, relational views, Contingency table, Supervised machine learning, Performance.