Integrating Decision Tree Learning into Inductive Databases

Élisa Fromont and Hendrik Blockeel

Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200 A, 3001 Heverlee, Belgium,
{elisa.fromont, hendrik.blockeel}@cs.kuleuven.be

Abstract. In inductive databases, there is no conceptual difference between data and the models describing the data: both can be stored and queried using some query language. The approach that adheres most strictly to this philosophy is probably the one proposed by the ADReM group from Antwerp: in that approach, models are stored in relational tables and queried using standard SQL. The approach has been described in detail for association rule discovery. In this work we study how decision tree induction could be integrated in that approach. We propose a representation format for decision trees similar to the one proposed earlier for association rules, and queryable using standard SQL; and we present a prototype system in which part of the needed functionality is implemented. In the process, we identify a number of important differences between discovery of global models (such as decision trees) and local models (such as association rules), which force us to re-evaluate the motivation for the approach.

1 Introduction

An Inductive DataBase (IDB) [1] is a database that contains not only data, but also generalisations (patterns and models) valid in the data. In an IDB, ordinary queries can be used to access and manipulate data, while inductive queries can be used to generate (mine), manipulate, and apply patterns.

Two approaches have been studied to represent and query patterns and models in IDBs. First, depending on the models that will be stored, special-purpose storage and query language can be created. In this context, several researchers have proposed extensions to the popular relational query language, SQL, as a natural way to express such mining queries. For example, in [2,3], the authors have presented some extensions to SQL especially designed for mining association rules. In [4,5] the authors extend this approach to other models such as classification rules but they do not give any clues about how to actually stored those models in the IDB. In [6], De Raedt has proposed a entirely new query language based on logic and especially suited for relational data.

The second approach consists of storing the patterns and models in a straightforward way, using the usual relational database tables provided by any Relational Database Management System (RDBMS) and the standard SQL language

to represent, store and query the new generalisations made on the data. This approach is being investigated by members of the ADReM group in Antwerp[1] for frequent itemsets and association rules mining; we will refer to it in the rest of this paper as "the ADReM approach". This approach has a number of advantages over other approaches with respect to extensibility and flexibility. In this paper we investigate whether and how it can also be used for learning global models such as decision trees, and to what extent its advantages carry over to this new setting.

In Section 2 we present the basic ideas behind the ADReM approach and show how they are applied in the context of association rule discovery. In Section 3, we discuss how the same approach could be used for decision tree learning; in particular, Subsection 3.5 describes a prototype that is being implemented and shows some examples of queries that can already be used. Section 4 presents the perspectives of this work and we conclude in Section 5.

2 The ADReM Approach to Association Rule Mining

The basic idea behind the ADReM approach is that models are stored in a relational database in the same way that other information is stored: as a collection of tuples in relations or views. This idea is applicable to a broad range of models, but up till now it has been studied mostly in the context of association rules [7]. Below, we briefly review the proposed representation for association rules, the conceptual view on association rule mining that it leads to, and some implementation issues. More information on this can be found in [8].

2.1 The Conceptual View

Sets

isid	item
i1	p4
i1	p6
i1	p5
...	...
i2	red
i2	green
...	...
i3	p4
i3	1
...	...

Supports

isid	support
i0	80
i1	60
i2	40
i3	80
...	...

Rules

rid	isida	isidc	isid	conf
r1	i2	i1	i5	60%
r2	i1	i3	i6	40%
...	...	...	...	...

Fig. 1. Storing association rules

[1] {toon.calders,bart.goethals,adriana.prado}@ua.ac.be

Consider a set of transactions D. The set is often represented as a table with one row per transaction and one boolean attribute per item, but conceptually it can also be represented as a binary relational table with, for each transaction, a set of tuples of the form (*tid,item*), where *tid* is the transaction identifier and *item* is an item name. The crucial difference between the first and second representation is that in the second, the item names are values instead of attributes (hence a query can return an item name as part of its result set). Note that we are talking about the conceptual representation here — how the transaction table is really implemented is not important.

Itemsets can be represented in a similar way. Figure 1 shows the ADReM representation of frequent itemsets and association rules in an IDB. The *Sets* table represents all itemsets. A unique identifier (*isid*) is associated to each itemset (we then write that itemset as $IS(isid)$) and, for each itemset of size n, there are n rows $(isid, item_j)_{1 \leq j \leq n}$ where $item_j$ is the j^{th} item of $IS(isid)$. The *Supports* table stores the *support* of each itemset. The *Rules* table stores the association rules computed. For each rule $X \Rightarrow Y$, there is a row $(rid, isida, isidc, isid, conf)$ in the IDB where *rid* is the association rule identifier, *isida* (resp. *isidc*) is the identifier of the itemset used in the antecedent (resp. consequent) of the rule, $IS(isid) = IS(isida) \cup IS(isidc)$ and $conf$ is the confidence of the rule.

With this representation, finding association rules subject to certain constraints can be done easily using an SQL query. For instance, the query

```
select rid
from rules r
where r.conf > 0.8 and
      r.isidc in (select isid from sets where item = ‘‘red’’)
```

finds all association rules with a confidence of at least 0.8 that contain the item "red" in the consequent of the rule.

2.2 The Implementation

Conceptually, the database has tables that contain all itemsets and all association rules. But in practice, obviously, the amount of such patterns can be huge and it may be impossible to store them all. This problem is solved by keeping these tables virtual. As far as the user is concerned, those tables or *virtual mining views* contain all the tuples needed to answer the user query. In reality, each time such a table is queried, a efficient data mining algorithm is triggered by the DBMS to populate those views just sufficiently for the DBMS to be able to answer the query.

More specifically, the procedure works as follows: given a query, an execution plan is created; on the highest level this is just a relational algebra tree with tables as leaves. Standard query optimisation procedures push projection and selection conditions as deeply down into the this tree as possible, thus reducing the size of intermediate results and making the overall computation more efficient. In the context we are discussing here, the leaves may be the result of a data

mining process, and the projection/selection conditions may be pushed further down into the data mining process. Calders et al. [8] describe this optimisation process in detail.

2.3 Advantages of the Approach

The ADReM approach has several advantages over other approaches to inductive querying. The main point is that the data mining processes become much more transparent. From the user's point of view, tables with itemsets and rules etc. exist and can be queried like any other table. How these tables are filled (what data mining algorithm is run, with what parameters, etc.) is transparent. The user does not need knowledge about the many different implementations that exist and when to use what implementation, nor does she need to familiarise herself with new special-purpose query languages. The whole approach is also much more declarative: the user specifies conditions on the models that should result from the mining process, not on the mining process itself.

In the light of all these advantages, it seems useful to try a similar approach for other data mining tasks as well. In this paper, we focus on decision tree induction.

3 Integration of Decision Tree Learning

A decision tree aims at classifying instances by sorting them down the tree from the root to a leaf node that provides the classification of the instance [9]. Each node in the tree specifies a test of some attributes of the instance, and each branch descending from that node corresponds to one of the possible values for these attributes. In this paper, for simplicity reasons, we focus on decision trees with boolean attributes. Each attribute can then be seen as an item in a transaction table and transactions are instances to classify. Note that due to the well-known correspondence between trees and rule sets, a tree can be represented as a set of association rules: with each leaf corresponds one association rule, with as antecedent the conditions on the path from the root to that leaf and as consequent the majority class in that leaf. However, while the representation with one rule per leaf is interesting for prediction purposes, the structure of the tree gives more information: e.g., the higher an attribute occurs in the tree, the more important it is for the prediction. Such information is lost if we represent a tree as a set of association rules with each rule corresponding to one leaf. We will therefore choose a slightly different representation.

In this section we first discuss the motivations for integrating decision trees into IDB and we propose a representation of decision trees that will enable the user to make queries using a large number of different constraints.

3.1 Motivations

To see the motivation behind using the ADReM approach for decision tree learning, consider the current practice in decision tree induction. Given a data set,

one runs a decision tree learning algorithm, e.g., C4.5 [10], and obtains one particular decision tree. It is difficult to characterise this decision tree in any other way than by describing the algorithm. The tree is generally not the most accurate tree on the training set, nor the smallest one, nor the one most likely to generalise well according to some criterion; all one can say is that the learning algorithm tends to give relatively small trees with relatively high accuracy. The algorithm usually has a number of parameters, the meaning of which can be described quite precisely in terms of how the algorithm works, but not in terms of the results it yields. In summary, it is difficult to describe exactly what conditions the output of a decision tree fulfils without referring to the algorithm.

This situation is quite different from discovery of association rules, where the user imposes some constraints on the rules to be found (typically confidence and support) and the algorithm yields the set of all rules fulfilling these conditions. A precise mathematical description of the result set is very easy to give, whereas a similar mathematical description of the tree returned by a decision tree learner is quite impossible to give.

Are the people using decision tree learners interested in having a precise specification of the properties of the tree they find? Aren't they just interested in finding some tree with good generalisation properties and good generalisation power, without being interested in exactly how this is defined? This may be often the case, but certainly not always. Many special versions of decision tree learners have been developed: some use a cost function on attributes and try to find trees that combine high accuracy with low cost of the attributes they contain [11]; some take different misclassification costs into account when building the tree [12]; some do not aim for the highest accuracy but for balanced precision-recall [13]; etc. The fact that researchers have developed such learning algorithms shows that users sometimes do have more specific desiderata than just high predictive accuracy.

By integrating decision tree learning into inductive databases, we hope to arrive at an approach for decision tree learning that is just as precise as association rule learning: the user specifies what kind of trees she wants, and the system looks for such trees.

Here are some queries the user might be interesting in:

1. find $\{T | size(T) < 8 \wedge acc(T) > 0.8 \wedge cost(T) < 70\}$
2. find one element in $\{T | size(t) < 8 \wedge acc(t) > 0.8 \wedge cost(t) < 70\}$
3. find $\{T | size(T) < 8 \wedge (\forall T' | size(T') < 8 \Rightarrow acc(T') < acc(T))\}$
4. find $\{T | T = (t(X, t(Y, l(+), l(-)), t(Z, l(\_), l(\_)),$
   $X \in [A, B, C], Y \in [D, E], acc(T) > 0.8\}$

In the first query, the user asks for all decision trees T of *size* less than 8 nodes, of *global accuracy* higher than 0.8 and of *cost* lower than 70 (assuming that each item has a given cost). To describe the tree of interest, other criteria such as the number of misclassified examples (*error*), the *accuracy* computed for a particular class, the *precision*, the *recall*, the area under the roc curve (*auc*) for two-class problems might also be interesting.

Since the user is interested in all the trees that fulfil his criteria, the query can not be answered by triggering a standard *greedy* algorithm. Such a query implies the use of a decision tree learner that can perform an *exhaustive* search in the search space of all possible trees that fulfil these constraints. The number of such trees might be huge and this query might not be tractable. Note that without constraints, the number of decision trees that can be constructed from a database containing d attributes and a possible values for each attribute has $\prod_{i=0}^{d-1}(d-1)^{a^i}$ as a lower bound. As in the association rules case presented in section 2 we assume that the queries are constrained enough so that a realistic number of models are looked for and stored. The kind of constraints required to satisfy this criterion is still an open question.

In the second query, the user asks for one tree that fulfils some criteria. This tree can normally be computed by a regular *greedy* tree learner, though for some greedy learners there may be no guarantee that they find a valid tree if one exists.

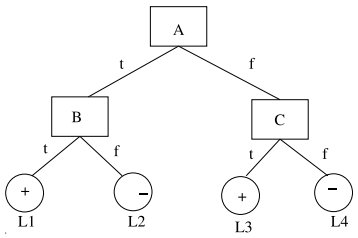
With the third query, the user is looking for the set of trees of size lower than 8 with maximal accuracy. Again, this means that the search space of trees of size lower than 8 must be exhaustively covered to ensure that the accuracy of the tree is maximal.

In the last query, the user gives *syntactic constraints* on the shape of the tree and on some attributes that must be used in the tree.

3.2 Representing Trees in a Relational Database

The virtual mining view that holds the predictive models should be precise enough to enable the user to ask SQL queries as easily as possible without having to design new keywords for the SQL language. Figure 2 shows the database framework we propose for integrating decision trees into IDB. We use the table presented in section 2 to represent the data. We assume in the following that all the data-dependent measures (such as accuracy) are referring to these data. The decision trees generated from the data D can be stored in the same database as D and as the possible association rules computed from D, by using the following schema:

1. The *Tree_sets* table is inspired by the *Sets* table created for association rules. We choose to represent a node of a tree by the itemset that characterises the examples that belong to this node. For instance, if the itemset is $A\overline{B}$, examples in this node must fulfil the criteria $A = true$ and $B = false$. In association rules, only the presence of certain items is required: there is no condition that specifies the absence of an item. To cope with association rules derived from trees such that the one corresponding to the leaf $L2$ of the tree given in the figure 2 ($A\overline{B} \Rightarrow -$), a *sign* attribute is thus added to the table to indicate wether the presence (1) or the absence (0) of an item is required.
   As in the *Sets* table, a unique identifier ($isid$) is associated to each itemset and, for each itemset of size n, there are n rows $(isid, item_j, sign_j)_{1 \leq j \leq n}$



Trees_charac

treeID	size	error	accuracy	auc	cost	...
T1	...	...	...	...	...	...
T2	...	...	...	...	...	...
...	...	...	...	...	...	...

Tree_sets

isid	item	sign
i0	$\emptyset$	1
i1	A	1
i2	A	1
i2	B	1
l1	A	1
l1	B	1
l1	+	1
i3	A	1
i3	B	0
l2	A	1
l2	−	1
l2	*B*	0
l3	...	...
l4	...	...
...	...	...

all_trees

treeID	isid	leaf
T1	i1	0
T1	i2	0
T1	L1	1
...	...	...
T1	L2	1
...	...	...
T1	L3	1
...	...	...
T1	L4	1
T2	...	...
T3	...	...
...	...	...

greedy_trees

treeID	isid	leaf
T1	i1	0
T1	i2	0
T1	L1	1
...	...	...
T1	L2	1
...	...	...
T1	L3	1
...	...	...
T1	L4	1

Fig. 2. Storing decision trees

where $item_j$ is the j^{th} item of the itemset identified by $isid$ and $sign_j$ is its sign. $i0$ stands for the empty itemset.

2. The *all_trees* and *greedy_trees* tables give a precise description of each tree in terms of the itemsets from *Tree_sets*. A unique identifier ($treeID$) is associated to each tree and each itemset corresponding to a node in this tree is associated to his $treeID$. A boolean attribute $leaf$ differentiates internal nodes of the tree ($leaf = 0$) from the leaves ($leaf = 1$). At each level k of the tree, nodes are composed by k-itemsets. The *all_trees* table is supposed to hold all possible trees, whereas the *greedy_trees* table holds an implementation-dependent subset of all trees (more specifically those trees that might be found by greedy learners under certain conditions and constraints).
3. The *Trees_charac* table gives all the characteristics of the tree the user might be interested in. For each tree identified by $treeID$ corresponds a row $(size, error, accuracy, cost, auc)$ that are the computed characteristics of the tree (see section 3.1).

3.3 Querying Decision Trees Using Virtual Views

The structure created is sufficient to make some interesting queries on the data, provided that the data mining algorithms connected to the database compute the different characteristics of the association rules or of the trees that hold in the IDB. The following queries are examples of what can be done on such database :

```
SELECT trees_charac.* FROM trees_charac, all_trees
WHERE trees_charac.treeID = all_trees.treeID AND
 accuracy > 0.8 and size < 8;
```

This query selects the characteristics of all trees that can be computed from the database with accuracy higher than 0.80 and size lower than 8.

```
SELECT treeID FROM trees_charac, greedy_trees
WHERE trees_charac.treeID = greedy_trees.treeID
and trees_charac.error < 10;
```

This query selects a greedy-computed tree with an error lower than 10.

```
SELECT trees_charac.* FROM trees_charac, all_trees
WHERE trees_charac.treeID = all_trees.treeID
AND accuracy = (select max(accuracy) from trees_charac);
```

This query selects the characteristics of the tree with maximum accuracy from all the possible computed trees.

```
SELECT treeID FROM greedy_trees, tree_sets
WHERE greedy_trees.isid = tree_sets.isid
AND (tree_sets.isid
     IN (select isid from tree_sets where item = ''A''));
```

This query selects a greedy-computed tree which contains the item "A".

3.4 Querying Decision Trees Using Itemsets

The framework is flexible enough to allow queries with constraints on metrics that were not included in the virtual view from the beginning. The user can create his own virtual mining view using information on the support of the itemsets. We illustrated this with the notions of accuracy and size.

The accuracy of a specific leaf in the tree can be computed from the support of the itemsets that belong to the leaf [14] using the formula (on figure 2):

$$acc(L_1) = \frac{support(+AB)}{support(AB)} \ldots acc(L_2) = \frac{support(-A\overline{B})}{support(A\overline{B})}.$$

The mean accuracy of each leaf is the global accuracy of the tree. This can be computed without any information on the actual structure of the tree using the

formula (on figure 2):

$$acc(T) = \frac{acc(L1) * support(AB) + acc(L2) * support(A\overline{B}) + \ldots}{support(\emptyset)}$$
$$= \frac{support(+AB) + support(-A\overline{B}) + \ldots}{support(\emptyset)}.$$

Some itemsets do not have any support associated with because they include "negative" item. In this case, some formula such as:

$$support(A\overline{B}-) = support(A-) - support(AB-)$$

can be used to compute the support of all itemsets from the support of the "positive" itemsets [15].

These formulas can be translated into the SQL language to compute all the characteristics in the *tree_charac* table. We consider that, as in Section 2, we have a Supports table that countains the support of all itemsets.

```
acc(T1)= SELECT SUM(Supports.support) /
(SELECT Supports.support FROM Supports
WHERE Supports.isid = ‘‘I0’’) as accuracy
FROM Supports, all_trees
WHERE Supports.isid = all_trees.isid
AND all_trees.treeID = T1
GROUP BY all_trees.treeID

size(T1) = SELECT COUNT(*) FROM all_trees
WHERE all_trees.treeID = T1
```

3.5 Implementation

An Apriori-like algorithm for association rule mining was connected to a standard Oracle Database by the ADReM group to use constraints such as the support of the itemsets, the confidence of the rules and the presence or absence of some item in the resulting association rules. We connected to the same system a decision tree learner named *Clus*. Clus is a predictive clustering tree learner developed by Jan Struyf that uses a standard recursive top-down induction algorithm to construct decision trees. First, a large tree is built based on the training data then the tree is pruned according to some user constraints. The constraints described by [16] were implemented in this generic and efficient system [17] so it currently supports constraints on the size of the tree (number of nodes), on the error of the tree and on the syntax of the tree. The error measure used for classification trees learning is the number of incorrectly predicted classes. The syntactic constraints allow the user to introduce expert knowledge in the tree by specifying a partial tree, i.e, a subtree including the root and so the most important attributes of the tree. Other constraints discussed in section 3.1 have to be implemented in the system. For the moment, queries such as the following one can be used:

```
SQL> select * from trees_charac where err < 8 and sz= 9;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      0  9     3     0,98
1 rows selected.


SQL>select * from trees_charac where err < 8 and sz <= 8;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      1  7     4    0,973
1 rows selected.

SQL> select * from trees_charac where sz< 4;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      2  3    50    0,667
1 rows selected.
```

All the trees computed with the different queries could be stored in a "log" table that can be queried just as easily. After the session above, this table would contain:

TREE_ID	SZ	ERROR	ACCURACY
0	9	3	0,98
1	7	4	0,973
2	3	50	0,667

4 Perspectives

There are many open problems related to the proposed approach. For instance, for efficiency reasons, the system should be able to look at the "log" table that contains the previously computed trees to check if the answer of the current query has not already been computed before triggering a data mining algorithm. If the user asks for all trees of size less than 8 and then later for all trees of size less than 6, the results computed from the first query should be reusable for the second query. The "log" table should then also contain the previously asked queries together with the computed trees, which raises the question of how to store the queries themselves in the database. This entire problem, called *interactive mining* because it refers to the reutilisation of queries posed within the same working session, has been investigated for association rules [18], but not yet for decision tree learning.

Another type of problem occurs if the database has been modified between two queries. Is it possible to use some previously computed predicted models to

compute more efficiently new predictive models from a modified database? This problem known has *incremental learning* has already been studied for decision trees [19] when a new example is added to the database.

These functionalities has to be integrated into the prototype along with the extension of the framework to multi-valued attributes.

Besides, as predictive models ultimately aim at predicting the class of new examples, it would be interesting to include that possibility in the IDB. This is currently non-trivial in our approach, it requires complicated queries.

Generally, the limitations of our approach with respect to what can be expressed, and how difficult it is to express it, are still unclear. With respect to query complexity, it may be useful to consider an extended relational model where trees are an abstract data type with a number of predefined operators, instead of being stored as sets of tuples.

Another perspective will be the integration of other predictive models such as *Bayesian Network* in the same IDB framework already designed for association rules and decision trees mining. The user might be interested in query such as "find the Bayesian network of size 3 with maximal probability". Again, a structure to store Bayesian networks has to be designed and algorithm than can build Bayesian networks under constraints has to be implemented.

5 Conclusion

In this paper we have studied how decision tree induction could be integrated in inductive databases following the ADReM approach. Considering only boolean attributes, the representation of trees in a relational database is quite similar to that of association rules, with this difference that the conjunctions describing nodes may have negated literals whereas itemsets only contain positive literals. A more important difference is that a decision tree learner typically returns one tree that is "optimal" in some not-very-precisely-defined way, whereas the IDB approach lends itself more easily to mining approaches that return all results fulfilling certain well-defined conditions. It is therefore useful to introduce a *greedy_trees* table in addition to the *all_trees* table, where queries to *greedy_trees* trigger execution of a standard tree learner and queries to *all_trees* trigger execution of an exhaustive tree learner. We have described a number of example queries that could be used, presented a preliminary implementation that handles such queries, and discussed open questions and perspectives of this work.

Acknowledgement

Hendrik Blockeel is a post-doctoral fellow of the Fund For Scientific Research of Flanders (FWO-Vlaanderen). This work is funded through the GOA project 2003/8, "Inductive Knowledge bases", and the FWO project "Foundations for inductive databases". The authors thank Jan Struyf, Sašo Džeroski and the ADReM group for many interesting discussions, and in particular Jan for his help with the Clus system and Adriana Prado for her help with the IDB prototype.

References

1. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Comm. Of The Acm **39** (1996) 58–64
2. Meo, R., Psaila, G., Ceri, S.: An extension to sql for mining association rules. Data Min. Knowl. Discov. **2** (1998) 195–224
3. Imielinski, T., Virmani, A.: Msql: A query language for database mining. Data Min. Knowl. Discov. **3** (1999) 373–408
4. Kramer, S., Aufschild, V., Hapfelmeier, A., Jarasch, A., Kessler, K., Reckow, S., Wicker, J., Richter, L.: Inductive databases in the relational model: The data as the bridge. In: KDID. (2005) 124–138
5. Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: DMQL: A data mining query language for relational databases. In: SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada (1996)
6. De Raedt, L.: A logical database mining query language. In Cussens, J., Frisch, A., eds.: ILP00. Volume 1866 of LNAI., SV (2000) 78–92
7. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, Morgan Kaufmann (1994) 487–499
8. Calders, T., Goethals, B., Prado, A.: Integrating pattern mining in relational databases. In: PKDD: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases. LNCS, Springer (2006) To appear
9. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
11. Turney, P.: Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. Journal of Artificial Intelligence Research **2** (1995) 369–409
12. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: Knowledge Discovery and Data Mining. (1999) 155–164
13. Xiaobing, W.: Knowledge representation and inductive learning with xml. In: WI '04: Proceedings of the Web Intelligence, IEEE/WIC/ACM International Conference on (WI'04), Washington, DC, USA, IEEE Computer Society (2004) 491–494
14. Pance, P., Dzeroski, S., Blockeel, H., Loskovska, S.: Predictive data mining using itemset frequencies. In: Zbornik 8. mednarodne multikonference Informacijska druzba. Ljubljana: Institut "Jozef Stefan", Informacijska druzba (2005) 224–227
15. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery. Volume 2431 of LNCS., Springer-Verlag (2002) 74–85
16. Garofalakis, M.N., Hyun, D., Rastogi, R., Shim, K.: Building decision trees with constraints. Data Min. Knowl. Discov. **7** (2003) 187–214
17. Struyf, J., Dzeroski, S.: Constraint based induction of multi-objective regression trees. In: KDID. (2005) 222–233
18. Goethals, B., den Bussche, J.V.: On supporting interactive association rule mining. In: Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery. Volume 1874 of LNCS., Springer (2000) 307–316
19. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning **4** (1989) 161–186