Mining Views: Database Views for Data Mining

Hendrik Blockeel[1], Toon Calders[2], Elisa Fromont[1],
Bart Goethals[3], and Adriana Prado[3]

[1] Katholieke Universiteit Leuven, Belgium
[2] Technische Universiteit Eindhoven, The Netherlands
[3] Universiteit Antwerpen, Belgium

Abstract. We propose a relational database model towards the integration of data mining into relational database systems, based on the so called virtual mining views. We show that several types of patterns and models over the data, such as itemsets, association rules, decision trees and clusterings, can be represented and queried using a unifying framework. We describe an algorithm to push constraints from SQL queries into the specific mining algorithms. Several examples of possible queries on these mining views, using the standard SQL query language, show the usefulness and elegance of this approach.

1 Introduction

Data mining is an iterative and interactive process. During the whole discovery process, typically, many different data mining tasks are performed, their results are combined, and possibly used as input for other data mining tasks. To support this knowledge discovery process, there is a need for integrating data mining with data storage and management. The concept of inductive databases (IDB) has been proposed as a means of achieving such integration [6].

In an IDB, one can not only query the data stored in the database, but also the patterns that are implicitly present in these data. The main advantages of integrating data mining into database systems are threefold: first of all, the data is mined where the data is: in the database. Hence, the need for transforming data into an appropriate format is completely removed. Second, in database systems, there is a clear separation between the logical and the physical level. This separation shields the user from the physical details, making the technology much more accessible for a non-specialist. Ideally, the user of an inductive database should not be involved with selecting the right algorithm and parameter setting, storage format of the patterns, etc., but should instead be able to specify, in a declarative way, the patterns in which he or she is interested. The third advantage of an inductive database is the flexibility of ad-hoc querying. In an inductive database, the user is not limited by the functionality offered by a limited set of tools. Instead, he or she can specify new types of patterns and constraints. Notice that data mining suites such as, e.g., Weka [19] and Yale [11] only share the first advantage of inductive databases by imposing one uniform data format for a group of algorithms.

In this paper, we focus our attention on determining how such an inductive database can be designed in practice. The solution proposed in this paper builds upon our preliminary work in [2, 3]. In contrast to the numerous proposals for data mining query languages [4, 7, 8, 10, 15, 17, 18], we propose to integrate data mining into database systems without extending the query language. Instead, we extend the database schema with new tables that contain, for instance, association rules, decision trees, or other descriptive or predictive models.

One might argue against this approach that tables containing all possible patterns and models over the data would in most cases be huge. These tables, however, are in fact implemented as views, called *virtual mining views*. Whenever a query is formulated that selects for instance association rules from these tables, this triggers a run of a data mining algorithm (e.g., Apriori [1]) that computes the result of the query, in exactly the same way that normal views in databases are only computed at query time, and only to the extent necessary for answering the query.

This querying approach naturally integrates constraint-based mining. Within the query, one can impose conditions on the kind of patterns that one wants to find. In many cases, these constraints can be pushed into the mining process.

The paper is organized as follows. Section 2 focuses on the related work. We introduce our new framework and the mining views it uses in Section 3. In Section 4 we illustrate how standard SQL queries on these views allow us to search for models fulfilling certain constraints or to apply a given model to a new dataset. In Section 5, we extend the constraint extraction algorithm from [2] for decision trees and clusterings. We conclude in Section 6.

2 Related Work

There already exist multiple proposals for extending a query language with some data mining primitives. The most well-known examples are the SQL-like operator MINE RULE of Meo et al. [10] for mining association rules, and the data mining query language DMQL by Han et al. [4]. In both studies, however, the language constructions only allow to specify the desired output, but this output is not integrated again into the database. Our proposal goes beyond this, by also allowing the results to be used as input for further data mining queries, as they are treated as regular database tables.

In Microsoft's Data Mining extensions (DMX) of SQL server [15], a classification model can be created. This model can be trained and used afterwards to give predictions, via the so-called prediction joins. However, this framework does not provide any operations other than browsing and prediction for manipulating the model, and there is no notion of composing mining operations in their framework. Although the philosophy behind the predictor join is somewhat related to our proposal, the work presented in this paper goes much further.

Siebes [14] argues in favour of making patterns and models first-class citizens, and suggests to extend for instance the relational algebra with operations on models. In our framework, predictive models are already first-class citizens in the relational algebra itself, as they are simply relations. As such, the operation of applying a predictive model M to an instance x simply corresponds to a selection and projection from M: $\pi_Y(\sigma_{X=x}(M))$; the composition of two predictive models is their join, etc.

The closest to the work presented in this paper are $\mathcal{LDL}^{++}$ [17] and ATLaS [9, 18, 20]. $\mathcal{LDL}^{++}$ and ATLaS are extensions of respectively $\mathcal{LDL}$ and SQL that add the ability of defining new *user defined aggregates* (UDAs), making them suitable for data mining. Especially ATLaS is very interesting with respect to our proposal, as it is also based on the principles of relational databases and query languages. In ATLaS, however, the query language is much more powerful (even Turing complete). In fact, ATLaS is rather a programming language based on SQL, enabling data mining operations, on top of relational databases. Hence, in ATLaS, the results of mining have to be encoded into the relational model, and subsequent queries of found patterns have to deal with decoding and encoding the found patterns. Also, the ATLaS query language is much less declarative, making it less attractive for query optimization.

As already pointed out in the introduction, the work presented here builds upon our own preliminary work on the integration of association rule mining and decision tree learning into database systems [2, 3]. This paper significantly improves upon these works in the following way. The representations of association rules and decision trees, as proposed in our earlier work, are fairly complex. In this work, we propose a new unifying representation that is more elegant and simpler than the originally proposed representations. It focuses more on the semantics of learned models rather than their structure, and as such allows us to handle conceptual models such as association rules, decision trees and clusterings in a general way. For instance, applying a model to classify a new example now amounts to a simple join operation, while involving much more complex queries using the previous representation. Furthermore, the constraint extraction algorithm of [2] is extended to support queries about predictive models as well.

3 Framework Representation

Given a table $T(A_1, \ldots, A_n)$, let $Dom(T) = Dom(A_1) \times \ldots \times Dom(A_n)$ denote the domain of T. We create a *Concept Table* $Concepts_T(Cid, A_1, \ldots, A_n)$, such that for every tuple t in T, there exist 2^n unique tuples $\{t'_1, \ldots, t'_{2^n}\}$ in $Concepts_T$ such that $t'_i.A_j = t.A_j$ or $t'_i.A_j = '?'$ for all $i \in [1, 2^n]$ and $j \in [1, n]$. We denote the special value $'?'$ as the *wildcard value* and assume it doesn't exist in the domain of any attribute. As each of the concepts can actually cover more than one tuple in T, a unique identifier Cid is associated to each concept.

A tuple, or *concept*, $(cid, a_1, \ldots, a_n) \in Concepts_T$ represents all tuples from $Dom(T)$ satisfying the condition $\bigwedge_{i|a_i \neq '?'} A_i = a_i$.

Figure 1 shows a data table for the classic *PlayTennis* example [12], together with a sample of its corresponding Concepts table.

PlayTennis

Day	Outlook	Temp	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
...	...	...	...	...	...

$Concepts_{PlayTennis}$

Cid	Day	Outlook	Temp	Humidity	Wind	Play
1	?	Sunny	?	High	?	No
2	?	Sunny	?	Normal	?	Yes
3	?	Overcast	?	?	?	Yes
4	?	Rain	?	?	Strong	No
5	?	Rain	?	?	Weak	Yes
6	?	?	?	?	?	?
...	...	...	...	...	...	...

Fig. 1. The PlayTennis data table and its corresponding Concepts table.

3.1 Representing models as sets of concepts

Given a data table T, and its corresponding Concepts table $Concepts_T$, we now explain how a variety of models can be represented using so called *virtual mining views* [2]. Although all mining views are defined over T, we omit the subscript T when it is clear from the context.

Itemsets and association rules Obviously, as itemsets in a relational database are conjunctions of attribute-value pairs, they can be represented as concepts. The result of frequent itemset mining can therefore be represented by a view $Sets(cid, supp)$. For each itemset, there is a tuple with *cid* the identifier of the itemset (concept) and *supp* its support. Also other attributes, such as χ^2 or any correlation measure, could be added to the view to describe the itemsets. Similarly, association rules can be represented by a view $Rules(rid, cida, cidc, cid, conf)$, where *rid* is the rule identifier, *cida* and *cidc* are the identifiers for the concepts representing the antecedent and the consequent of the rule respectively, *cid* is the union (disjunction) of these, and *conf* is the confidence of the rule. Again, many other attributes, such as lift, conviction, or gini index, could be added to describe the rules.

Predictive models In association rule discovery, results typically describe the dataset itself, but in inductive learning, one is interested in building from the training set a model of a broader population, from which the training set is a representative sample. Therefore, it is useful to distinguish T, the table from which we learn (the training set), $Dom(T)$, the domain of that table (the set of all conceivable instances), and P, the actual population from which T is a random sample. P may be a strict subset of $Dom(T)$: not every conceivable tuple may exist in the real world. For instance, if $T(Gender, Age, Pregnant?)$ is the set of all patients currently in some hospital, which is a subset of the set P of all possible patients, then $P \subset Dom(T)$: a tuple with Gender=male and Pregnant?=true would be in $Dom(T)$ but not in P.

From this point of view, we can describe inductive learning (whether it is descriptive or predictive) as deriving P from T. Generally, P can be described as a probability distribution over $Dom(T)$, but here we will focus on the simpler case where P is a subset of $Dom(T)$. P can then be represented in tabular form, using exactly the same schema as T, since both are subsets of $Dom(T)$.

Trees

treeId	cid
M1	1
M1	2
M1	3
M1	4
M1	5

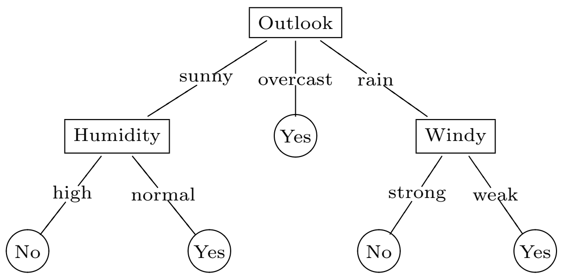


Fig. 2. Decision Tree built to predict the attribute Play.

In practice, machine learning systems would obviously learn P not in tabular format but, e.g., in the form of a decision tree. Such a tree represents a function from some attributes of T to a target attribute; this function is a relation, it is this relation that we call P.

Note that the term "model" may be used for both the representation of a model and its semantics (the relation it represents). In this text, we will use both meanings; it will usually be clear from the context what is meant.

We will now use concepts from the *Concepts* table to describe the semantics, and where possible also part of the structure, of the models learned. For instance, we represent all decision trees that can be learned from T, by the view *Trees*(*treeId*, *cid*). A unique identifier *treeId* is associated to each tree and each of the trees is described using a set of concepts (each concept describing one leaf). Figure 2 shows a decision tree built to predict the attribute Play using all other attributes in the data table, and its representation in the mining view Trees, using the Concepts table from Figure 1. If the user wants to build a tree from a subset of the attributes of the data table, he or she should first create a view on the data table that contains exactly the attributes he or she is interested in; a Concepts table related to this view can then be used to describe the trees.

Additionally, a view representing several characteristics of a tree learned for one specific target attribute A_i is added: *Trees_Charac_*A_i (*treeId*, *accuracy*, *size*, *cost*, ...). For every tree, there is a tuple with a tree identifier *treeId* and its corresponding characteristics.

Clustering In unsupervised learning (clustering), we can see $T(A_1, \ldots, A_n)$ as a projection of a relation $T'(A_1, \ldots, A_n, C)$ onto $A_1 \ldots A_n$ (the target attribute, indicating the class or cluster of each instance, is unobserved). P then has one more attribute than T, and can be seen as a predictive model over $Dom(T')$.

In extensional clustering, where the clusters are assumed to contain just the elements from the training set, P can be considered to have as many tuples as T. In intentional clustering, P would generalize over T in a similar way than predictive models do; i.e., it also assigns previously unseen instances to a cluster.

All possible clusterings that can be learned from T are represented in the view *Clusterings*(*clusId*, *clId*) and all clusters belonging to all clusterings are represented in the view *Clusters*(*clId*, *cid*). A unique identifier *clusId* is associated to each clustering and each of the clusterings is described by a set of clusters. A unique identifier *clId* is associated to each cluster and each of the clusters is described by a set of concepts.

Again, a view representing the characteristics of all clusterings is added: *Clusterings_Charac*(*clusId*, *size*, ...), with *size* the number of clusters. Of course other attributes could be added to this view.

Note that since the Concepts table has a finite number of elements (depending on the data table), the number of partitions (for clustering) as well as the number of trees that can be described using these concepts is also finite.

3.2 Discussion

The framework proposed in this paper is conceptually very different from the ones given in [2, 3] since it focuses on the representation of the semantics of the models (the function they represent) rather than on its structure (although, through the use of wild cards, limited information about the model structure is still available). Advantages of this approach are that it offers a unifying framework for all models, and that certain operations, such as using the model for prediction, become easier. On the contrary, queries about the structure, such as asking which attribute is at the root of a learned tree, become cumbersome. But, we believe that having a representation that is both suited for representing the complete structure and the semantics of the models is unrealistic. Descriptive or predictive models can be represented in many different formats (for example the ones proposed in [2, 3]), and for all these formats, views describing the model structure should be designed separately from the framework proposed here, and according to the needs and the preferences of the user.

The views representing the characteristics of a model provide information that may or may not be derivable from the tables that represent the models themselves. For instance, one could extend them with a full description of a decision tree (not just its set of leaves, as described by the Trees table). Including redundant information in the characteristics tables may simplify the formulation of constraints as queries.

In our current implementation, the identifiers in the mining views are "system-generated" values that have no meaning in the real world. Only equality of these identifiers within a query is well-defined; the same concept in different queries may have different identifiers, and vice versa. To avoid this, a kind of canonical encoding for (sets of) concepts would be needed, which is easy if the set of all concepts is known in advance but not when it depends dynamically on the extension of the relation, as is the case with our definition.

4 Model Querying

In this section, we give some concrete examples of common data mining tasks that can be expressed with SQL queries over the mining views. Compared to [2, 3], the new constructs introduced in this paper simplify the expression of queries for prediction and allow for a more declarative description of constraints on the desired models.

4.1 Prediction

In [3], to support decision trees in a relational database, the structure of decision trees was coded in a relational table. To predict the class of a new example, a query has to be written that explicitly expresses, almost in a procedural way, how the tree stored in the relation needs to be used. In our framework, a predictive model is stored as a set of concepts, capturing instead the semantics of the model. In order to classify a new example using one or more of the learned classifiers, one simply looks up the concept that covers the new example. More generally, if we have a test set S, all predictions of the examples in S are obtained by equi-joining S with the semantic representation of the classifier. As the concepts table is just a compact representation of this semantic view, we join S to *Concepts* using a variant of the equi-join that requires that either the values are equal, or there is a wild card.

Consider the classic *PlayTennis* example. The following query predicts the attribute Play for all unclassified examples in table *Test_Set*, considering all possible decision trees of size ≤ 5 in table Trees.

Test_Set

Day	Outlook	Temp	Humidity	Wind
D7	Sunny	Hot	High	Weak
D8	Rain	Hot	High	Strong
D9	Overcast	Hot	High	Weak
D10	Overcast	Mild	High	Weak
D11	Overcast	Cool	Normal	Weak
D12	Sunny	Cool	High	Strong

```
select T.treeId, S.*, C.Play
from Test_Set S,
     Trees T,
     Concepts C,
     Trees_Charac_Play D
where  T.cid = C.cid
 and  (S.Outlook = C.Outlook or C.Outlook ='?')
 and  (S.Temp = C.Temp or C.Temp = '?')
 and  (S.Humidity = C.Humidity or C.Humidity='?')
 and  (S.Wind = C.Wind or C.Wind='?')
 and  T.treeId = D.treeId
 and  D.size <= 5
```

4.2 Constraints

In this section, we discuss how typical constraints on association rules, decision trees or clusterings can be formulated as part of an SQL query. In our framework, these constraints can be expressed elegantly and more declaratively than in previous proposals.

For association rules, we consider constraints such as minimal and maximal support, minimal and maximal confidence, plus the constraints that a certain item must be in the antecedent, in the consequent, and boolean combinations of these. For decision trees, we consider the constraints size and accuracy. In addition to these, we also consider constraints posed on the concepts that describe the trees. For clusterings, we consider their size (number of clusters) and the popular constraints must-link (two instances must be in the same cluster) and cannot-link (two instances must not be in the same cluster) [16]. Next to these well-known constraints, in our approach, the user has also the ability to come up with new, ad-hoc constraints. In contrast, other proposals in the literature

```
      (A)
select R.rid,
       C1.*, C2.*,
       R.conf
from Sets S,
     Rules R,
     Concepts C1,
     Concepts C2
where R.cid   = S.cid
  and C1.cid  = R.cida
  and C2.cid  = R.cidc
  and S.supp  >= 30
  and R.conf  >= 80
```

```
      (B)
select T.*
from Trees_charac_Play T
where T.accuracy =
 (select max(accuracy)
  from Trees_Charac_Play T1
  and T1.size <= 5)
and T.size <= 5
```

```
        (C)
select T1.treeId,
       C1.*, C2.*
from Trees T1,
     Trees T2,
     Concepts C1,
     Concepts C2,
     Trees_Charac_Play D
where T1.treeId = T2.treeId
  and T1.cid    = C1.cid
  and C1.Outlook= 'Sunny'
  and T2.cid    = C2.cid
  and C2.Wind   = 'Weak'
  and T1.treeId = D.treeId
  and D.size     <= 5
  and D.accuracy >= 0.8
```

```
       (D)
select T.treeId, C.*
from Trees T,
     Concepts C
where T.cid = C.cid
  and not exists
   (select *
    from Concepts C1
    where C1.cid = C.cid
       and C1.Temp = '?')
```

```
       (E)
select C.clusId
from Clustering C
     Clusters Cl1,
     Clusters Cl2,
     I_Concepts I1,
     I_Concepts I2
 where I1.Day  = 'D1'
   and I2.Day  = 'D2'
   and C.clId  = Cl1.clId
   and Cl1.clId= Cl2.clId
   and Cl1.cid = I1.cid
   and Cl2.cid = I2.cid
```

```
        (F)
select C1.clusId
from Clustering C1,
     Clustering C2,
     Clusters Cl1
     Clusters Cl2,
     I_Concepts I1
     I_Concepts I2,
 where I1.Day   = 'D1'
   and I2.Day   = 'D2'
   and Cl1.cid  = I1.cid
   and Cl2.cid  = I2.cid
   and C1.clusId=C2.clusId
   and C1.clId  = Cl1.clId
   and C2.clId  = Cl2.clId
   and Cl1.clId <> Cl2.clId
```

Fig. 3. Example mining queries.

that extend the query language, in general do not allow this flexibility; only those constraints the language designer explicitly added to the language can be expressed.

Consider, again, the table *PlayTennis*. Figure 3 illustrates several mining queries that can be posed in our inductive database. Some constraints can be directly imposed using the tables *Sets*, *Rules*, *Trees_Charac* or *Clusterings_Charac* as shown in queries (A), (B) and (C). Query (A) asks for association rules having support of at least 30 and confidence of at least 80%. Query (B) selects decision trees having the attribute *Play* as the target attribute, having maximal accuracy among all possible decision trees of size ≤ 5. Query (C) asks for decision trees having a test on "Outlook=Sunny" and on "Wind=Weak", with a size of at most 5 and an accuracy of at least 80%.

Some constraints can also be imposed independently from the tables with the characteristics. For example, Query (D) asks for decision trees where the attribute Temp is never a wild card.

The popular must-link and cannot-link constraints, for clusterings, can also be expressed with SQL queries in our approach. Queries (E) and (F), respectively, are examples of how the user can formulate such constraints. In both queries, $I\_Concepts(Day, cid)$ is a view associating every instance in the data table with its covering concepts, which can be easily created by the user. Hence, query

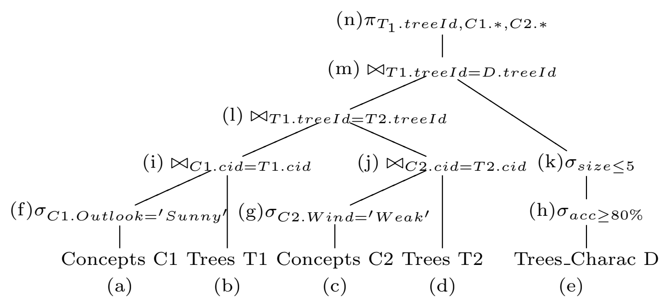

(n)$\pi_{T_1.treeId,C1.*,C2.*}$
(m) $\bowtie_{T1.treeId=D.treeId}$
(l) $\bowtie_{T1.treeId=T2.treeId}$
(i) $\bowtie_{C1.cid=T1.cid}$
(j) $\bowtie_{C2.cid=T2.cid}$
(k)$\sigma_{size\leq 5}$
(f)$\sigma_{C1.Outlook='Sunny'}$
(g)$\sigma_{C2.Wind='Weak'}$
(h)$\sigma_{acc\geq 80\%}$
Concepts C1
Trees T1
Concepts C2
Trees T2
Trees_Charac D
(a)
(b)
(c)
(d)
(e)

Fig. 4. An equivalent relational algebra for query (C) in Figure 3.

(E) asks for clusterings in which the instances "D1" and "D2" are in the same cluster, that is, in which both instances are covered by concepts describing the same cluster. Query (F) is formulated by using the opposite reasoning.

Hence, many well-known and common constraints can be expressed quite naturally in our model. In particular, queries that impose semantic restrictions, such as queries for prediction, or semantic constraints, such as must-link and cannot-link constraints, can be expressed more declaratively in our new framework. This more declarative nature of the queries also improves the ability to extract and exploit constraints in the queries imposed by the user for making the underlying mining operations more efficient.

5 Constraints Extraction

In the proposed framework, the tables are virtual. This means that for answering a query involving one or more of these views, we first need to instantiate them with the information needed by the query. Obviously, adding all concepts to the *Concepts* table, all trees to the *Trees* table, etc. is infeasible. However, since we expect the user to give a reasonable amount of constraints, only a subset of all tuples will be needed to answer the query correctly. In this section, we propose an algorithm that extracts constraints on the models needed to be mined from a given SQL query over the virtual views. These constraints can then be exploited by the data mining algorithm that is going to compute the result of the query. Consider for example query (C) in Figure 3. In order to answer that query, not all decision trees need to be mined, but only those with a size of at most 5, an accuracy of at least 80%, and node tests "Outlook=Sunny" and "Wind=Weak." In this context, the task of the constraint extraction algorithm is to extract these constraints from the query, such that they can be exploited by the tree inducer that has to be triggered to compute the result. Our constraint extraction algorithm finds constraints for all tables in the from-clause of the query, hence restricting the tuples required in the views to answer the query.

Algorithm The algorithm for extracting the constraints is an extension of the algorithm presented in [2], which only extracts constraints for itemset and association rule queries. It starts by building an equivalent relational algebra tree. For the example query (C), the tree is given in Figure 4. Notice that the views in the from-clause of the query correspond to the leaves of the expression tree. The idea is to find a constraint for each of those nodes while traversing the expression tree bottom-up. During the traversal, annotations expressing the constraints are computed for each of the nodes, based on the relational algebra operator in that node and the annotations of its children in the tree. For a node n, the annotation expresses the set of tuples needed in order to answer the sub-query rooted at that node. Hence, the annotation for the root node is the one we are looking for.

Annotations Consider, e.g., node (f) in the example query given in Figure 4. The sub-query associated with this node asks for all tuples in the table *Concepts* $C1$ with "Outlook=Sunny". The annotation for this node is:

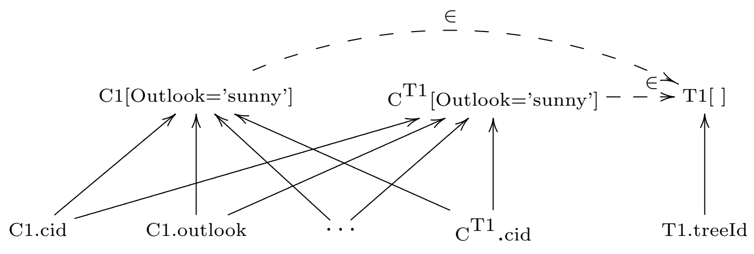
$$\begin{tikzcd}
 & \text{C1[Outlook='Sunny']} & & \\
\text{C1.cid} \arrow[ur] & \text{C1.Outlook} \arrow[u] & \text{C1.Wind} \arrow[ul] & \cdots \arrow[ull]
\end{tikzcd}$$

The top line in this annotation gives the views needed in order to answer the sub-query rooted at node (f). Between the square brackets a constraint on the tuples needed from this view is given. In the example node (f), we only need those tuples in the view *Concepts* $C1$ that satisfy "Outlook='Sunny' ". The bottom line in the annotation lists all attribute names of the sub-query. The arrows represent the view they originate from. Notice that an attribute can originate from more than one view. This occurs, e.g., when two relations were joined on this attribute. If, later on, an attribute is used in the condition of a selection in the tree, the constraint(s) of the view(s) from which that attribute originates will be updated.

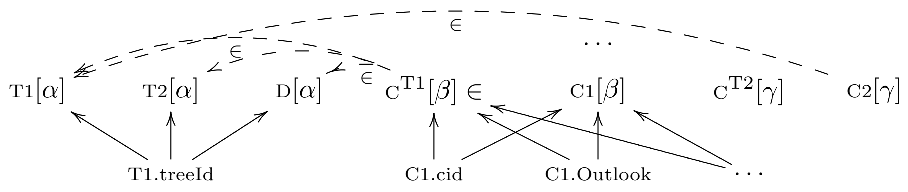
The construction procedure of the annotations needed for association rules and itemsets presented in [2] applies to our new framework as well. For decision trees and clusterings, however, the rules for constructing the annotations are more complicated. Indeed, while itemsets are described by a single concept and association rules are described by two concepts only (i.e., the antecedent and the consequent), decision trees and clusterings are both described by a priori unknown sets of concepts. In fact, a tuple $(treeId, cid)$ of the view $Trees$ represents two objects: the tree identified by $treeId$ and the concept with identifier cid belonging to it. In our annotation, this is expressed by two variables, one for the tree and one for the concept. They are connected with a dashed arrow from the concept to the tree, expressing that this concept belongs to that tree. For example, the annotation of nodes (b) and (d) are respectively:

$$\begin{tikzcd}
\text{C}^{\text{T1}}[\;] \arrow[r, dashed, bend left, "\in"] & \text{T1}[\;] & \text{C}^{\text{T2}}[\;] \arrow[r, dashed, bend left, "\in"] & \text{T2}[\;] \\
\text{C}^{\text{T1}}\text{.cid} \arrow[u] & \text{T1.treeId} \arrow[u] & \text{C}^{\text{T2}}\text{.cid} \arrow[u] & \text{T2.treeId} \arrow[u]
\end{tikzcd}$$

In node (i), $C1$ and $T1$ are joined on attribute *cid*, the annotation of node (i) will express that both concepts $C1$ and C^{T1} belong to the tree $T1$ and that they are actually the same concept. Every attribute of $C1$ also belongs to C^{T1} (and vice versa) and they inherit the constraints between the squared brackets. Hence, the annotation of (i) is as follows:



The annotations for the other nodes are built in the same way, resulting in the following:



$$\alpha = (acc \geq 80\% \;\wedge\; size \leq 5), \beta = (\text{Outlook = 'Sunny'}), \gamma = (\text{Wind = 'Weak'})$$

To ease the readability, we did not draw all ∈-arrows, but, actually, from every variable representing a concept, there is an arrow to $T1$, $T2$, and D. From this final annotation, finding the final constraints on the mining views is straightforward: for example, for the view $T1$, we see that not all possible trees are needed, but only those that satisfy condition α, and, also, have concepts that satisfy conditions β and γ. These constraints can be exploited directly by a tree inducer. Also for the other views, constraints can be extracted. For the Concepts table, e.g., it can be derived that not every concept should be there, but only concepts belonging to trees in $T1$.

6 Conclusion and Future Work

In this paper, we proposed a framework towards the integration of (constraint-based) data mining in a relational database, based on the so-called *mining views*. A mining view is a virtual table that contains models of the data. The main advantage over earlier proposals is that our schema elegantly covers a wider variety of models in a more uniform way, and that it makes it easier to define meaningful operations (e.g., predictions of new examples). A key component of the proposed approach is the use of the virtual *Concepts* table, which contains all conjunctive concepts definable over the relation that is being mined. We have illustrated how association rules, decision trees and clusterings can uniformly be expressed in terms of this *Concepts table*. Furthermore, we have shown how to formulate constraints in a query, using this structure, and how to automatically extract constraints from a given query for these different models.

As a proof of concept, the ideas presented in this work have been implemented into PostgreSQL [5]. The system is currently linked to algorithms for assocation rule discovery and exhaustive decision tree learning [3] (an exhaustive clustering algorithm is not yet available). The prototype shows promising results, for instance: for the UCI dataset *ZOO* [13], a query for all association rules with constraints support$\geq$30 and confidence$\geq$80% is executed in 2.3 seconds; querying for all decision trees with size$\leq$5 (without further constraints) takes 3.6 seconds.

We identify three directions for further work. First, in the current system, if the database is modified between two queries, the efficiency of the system could still be improved by investigating how to reuse the previously computed predictive models in order to compute new predictive models for the modified database. Second, it might also be interesting to reuse the results of related queries posed within the same working session. Finally, the schema described so far covers association rules, decision trees and clusterings. An obvious direction for further research is to extend it with other models.

Acknowledgements Hendrik Blockeel is a post-doctoral fellow from the Research Foundation – Flanders (FWO-Vlaanderen). This research was funded through K.U.Leuven GOA project 2003/8, "Inductive Knowledge bases", FWO project "Foundations for inductive databases" and the EU project "Inductive Queries for Mining Patterns and Models".

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
2. T. Calders, B. Goethals, and A. Prado. Integrating pattern mining in relational databases. In *Proc. 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, LNCS, pages 454–461. Springer, 2006.
3. E. Fromont and H. Blockeel. Integrating decision tree learning into inductive databases. In *ECML/PKDD-2006 International Workshop on Knowledge Discovery in Inductive Databases (KDID)*, pages 59–70, 2006.
4. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, 1996.
5. http://www.postgresql.org/
6. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The Acm*, 39:58–64, 1996.

7. T. Imielinski and A. Virmani. Msql: A query language for database mining. *Data Min. Knowl. Discov.*, 3(4):373–408, 1999.
8. S. Kramer, V. Aufschild, A. Hapfelmeier, A. Jarasch, K. Kessler, S. Reckow, J. Wicker, and L. Ritcher. Inductive Databases in the Relational Model: the Data is the Bridge. In *ECML/PKDD-2005 International Workshop on Knowledge Discovery in Inductive Databases (KDID)*, pages 124–138, 2005.
9. Y.-N. Law, H. Wang, and C. Zaniolo. Query languages and data models for database sequences and data streams. In *Proc. VLDB Int. Conf. Very Large Data Bases*, pages 492–503, San Francisco, CA, USA, 2004.
10. R. Meo, G. Psaila, and S. Ceri. An extension to sql for mining association rules. *Data Min. Knowl. Discov.*, 2(2):195–224, 1998.
11. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks, 2006.
12. T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
13. D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.
14. A. Siebes. Data Mining in Inductive Databases. In *ECML/PKDD-2005 International Workshop on Knowledge Discovery in Inductive Databases (KDID)*, pages 1–23, 2005.
15. Z. H. Tang and J. MacLennan. *Data Mining with SQL Server 2005*. John Wiley & Sons, 2005.
16. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the 17th Int. Conference on Machine Learning*, pages 1103–1110, 2000.
17. H. Wang and C. Zaniolo. Nonmonotonic reasoning in ldl++. *Logic-based artificial intelligence*, pages 523–544, 2001.
18. H. Wang and C. Zaniolo. Atlas: A native extension of sql for data mining. In *SIAM Intl. Conf. Data Mining*, pages 130–144, 2003.
19. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition, 2005.
20. C. Zaniolo. Mining databases and data streams with query languages and rules. In *ECML/PKDD-2005 International Workshop on Knowledge Discovery in Inductive Databases (KDID)*, pages 24–37, 2005.