Mining Views: Database Views for Data Mining

[redacted]

***Abstract*— We present a system towards the integration of data mining into relational databases. To this end, a relational database model is proposed, 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 and decision trees, can be represented and queried using a unifying framework.**

I. Motivation

Data mining is not a one-shot activity, but rather 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 [1].

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 are mined where they are located: 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 best algorithms, the parameter settings, the 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 IDB is the flexibility of ad-hoc querying. That is, the user can specify new types of constraints and query the patterns and models in combination with the data itself and so forth. Notice that the functionality of an inductive database goes far beyond that of data mining suites such as, e.g., Weka [2] and Yale [3]. These systems typically only share the first advantage of inductive databases by imposing one uniform data format for a group of algorithms.

In this work, we focus our attention on determining how such an inductive database can be designed in practice.

II. Description of the System

The system proposed in this paper builds upon our preliminary work in [4], [5]. In contrast to the numerous proposals for data mining query languages, we propose to integrate data mining into database systems without extending the query language. Instead, we extend the database schema with new tables containing, for instance, association rules, decision trees, or other descriptive or predictive models. As far as the user is concerned, these tables contain all possible patterns, trees, and models that can be learned over the data. Of course, such tables would in most cases be huge. Therefore, these tables are in fact implemented as views, called *virtual mining views*.

Whenever a query is formulated selecting for instance association rules from these tables, a run of a data mining algorithm is triggered (e.g., Apriori [6]) 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. The complete system is illustrated in Figure 1. When the user formulates his or her mining query, the parser is invoked by the DBMS creating an equivalent relational algebra expression. At this point, the expression is processed by the *Mining Extension* which extracts from the query the constraints that can be pushed into the data mining algorithms. The output of these algorithms is then materialized in the virtual mining views. After the materialization, the work-flow of the DBMS continues as usual and, as a result, the query is executed as if all patterns and models are stored in the database. Observe that this system can possibly cover every mining techniques whose output can be completely stored in relational tables.

This approach also integrates constraint-based mining in a natural way. Within a query, one can impose conditions on the kind of patterns or models that one wants to find. In many cases, these constraints can be pushed into the mining process. In [4], Calders et al. present an algorithm that extracts from a query a set of constraints relevant for association rules to be pushed into the mining algorithm. In this way, not all possible patterns or models need to be generated, but only those required to evaluate the query correctly as if all possible patterns or models were stored. We have extended

978-1-4244-1837-4/08/$25.00 © 2008 IEEE

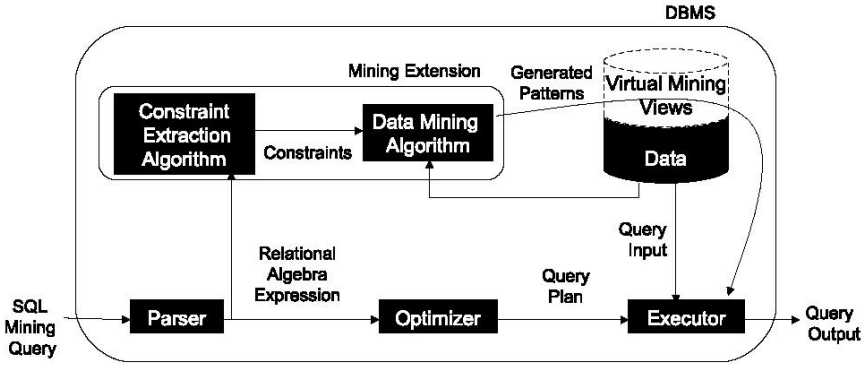

Fig. 1. The integration of data mining into a DBMS

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
...	...	...	...	...	...

PlayTennis_Concepts

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. 2. The PlayTennis data table and its corresponding *Concepts* table.

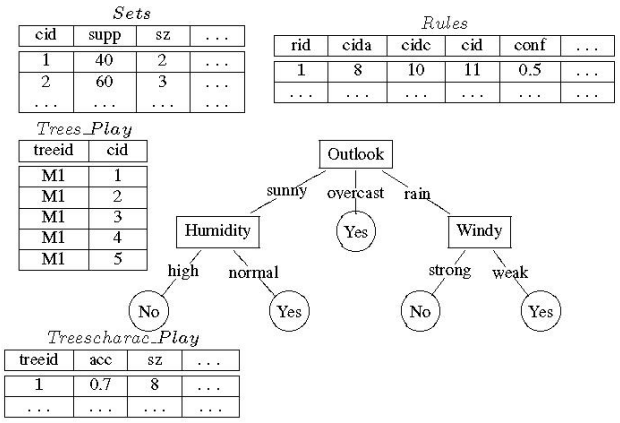

Fig. 3. Framework for an Inductive Database

this constraint extraction algorithm to extract constraints from queries over decision trees. The user can refer to [7] for more details on the algorithm.

III. 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 *Concepts* table $T\_Concepts(cid, A_1, \ldots, A_n)$, 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*, $(cid, a_1, \ldots, a_n) \in T\_Concepts$ represents all tuples from $Dom(T)$ satisfying the condition $\bigwedge_{i|a_i \neq '?'} A_i = a_i$.

Figure 2 shows a data table for the classic *PlayTennis* example [8], together with a sample of its corresponding *Concepts* table.

A. Representing models as sets of concepts

In this section, we explain how a variety of models can be represented using *virtual mining views*. We assume from now on that the *Concepts* table presented above is implemented as a virtual mining view. In this context, given a data table T and its corresponding virtual mining view $T\_Concepts$, the virtual mining views for itemsets, association rules and decision trees, based on $T\_Concepts$, are given in Figure 3. Although all virtual mining views are defined over T, in this text we omit the prefix T when it is clear from the context.

Notice again that Figure 3 only gives one possible instantiation of the proposed approach. As pointed out in Section II, virtual mining views for other mining techniques can be added to the system.

The proposed framework can be described as follows:

1) Itemsets and Association Rules: The result of frequent itemset mining can therefore be represented by a view $Sets(cid, supp, sz)$. For each itemset, there is a tuple with cid the identifier of the itemset (concept), $supp$ its support, and sz its size. Other attributes, such as χ^2 or any correlation measure, could also 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. Many other attributes, such as lift, conviction, or gini index, could be added to describe the rules.

2) Decision trees: We represent all decision trees that can be learned from T for one specific target attribute A_i by the view $Trees\_A_i(treeid, cid)$. A unique identifier $treeid$ is associated with each decision tree and each of the decision trees is described using a set of concepts (there is at least one concept describing one leaf). If the user prefers to build a decision tree from only a subset of the attributes, he should first create a view on the data table containing exactly those attributes, such that the mining view $Concepts$ associated with this view can then be used to describe the tree. We added a view $Treescharac\_A_i(treeid, acc, sz)$ to represent several characteristics of a decision tree learned for one specific target attribute A_i. For every tree, there is a tuple with a tree identifier $treeid$, acc its accuracy and sz its size (number of nodes). Again, other attributes could be added to describe the decision trees. Figure 3 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 first five concepts of the mining view $Concepts$ from Figure 2.

IV. Model Querying

In this section, we give some concrete examples of common data mining tasks that can be expressed with SQL queries over the virtual mining views. These examples support our claim that the virtual mining views provide an elegant way to incorporate data mining capacities to database systems without changing the query language.

A. Prediction

In order to classify a new example using one or more of the learned decision trees, 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 decision tree given in the virtual mining view $Concepts$. We join S to $Concepts$ using a variant of the equi-join that requires that either the values are equal, or there is a wildcard value.

Consider the *PlayTennis* example of Figure 2. Figure 4 illustrates a query that predicts the attribute Play for all unclassified examples in table $Test\_Set$, considering all possible decision trees of size ≤ 5.

B. Constraints

For itemsets and association rules, we consider constraints such as minimal and maximal size, minimal and maximal support, minimal and maximal confidence, plus the constraints that a certain item must be in the antecedent, in the consequent of the rules, and Boolean combinations of these. For decision trees, we consider constraints on size and accuracy. In addition to these, we also consider constraints posed on the concepts that describe the trees. Next to these well-known constraints, in our approach, the user has also the ability to come up with new, ad-hoc constraints.

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_Play T,
     Concepts C,
     Treescharac_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.sz       <= 5
```

Fig. 4. Prediction

```
      (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
  and S.sz    <= 4
```

```
      (B)
select T.treeid, C.*
from Treescharac_Play D,
     Trees_Play T,
     Concepts C
where T.cid = C.cid
 and  T.treeid = D.treeid
 and  D.sz <= 5
 and  D.acc =
  (select max(acc)
    from Treescharac_Play D1
    where D1.sz <= 5)
```

```
      (C)
select distinct D.*
from Trees_Play T1,
     Trees_Play T2,
     Treescharac_Play D,
     Concepts C1,
     Concepts C2
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.sz      <= 5
  and D.acc     >= 70
```

```
      (D)
select distinct D.*
from Trees_Play T,
     Treescharac_Play D,
     Concepts C
where T.cid    = C.cid
  and T.treeid = D.treeid
  and D.sz  <= 5
  and not exists
   (select *
    from Trees_Play T1,
       Treescharac_Play D1,
         Concepts C1
    where T1.cid    = C1.cid
      and T1.treeid = D1.treeid
      and D1.sz  <= 5
      and C1.cid = C.cid
      and C1.Temp = '?')
```

Fig. 5. Example mining queries

Figure 5 illustrates several mining queries that can be posed in our inductive database shown in Figures 2 and 3. Some constraints can be directly imposed using the tables $Sets$, $Rules$ and $Treescharac$ as shown in queries (A) and (B). Query (A) asks for association rules having support of at least 30, confidence of at least 80%, and size of at most 4. Query (B) selects decision trees having the attribute $Play$ as the target attribute and having maximal accuracy among all possible decision trees of size ≤ 5. The user can also constrain the concepts by which the models are described. For example, query (C) asks for decision trees having a test on "Outlook=Sunny" and on "Wind=Weak", while query (D) asks for trees where the attribute Temp is never a wildcard

TABLE I
EXECUTION TIMES FOR THE QUERIES IN FIGURE 5

#	Time (s)	#Concepts	#Rules	Output (rows)
(A)	0,60	2397	11525	11525
#	Time (s)	#Concepts	#Trees	Output (rows)
(B)	4,35	652	439	31
(C)	1,36	76	33	2
(D)	4,59	652	439	36

value (the attribute is present in every leaf of the tree).

Hence, many well-known and common constraints can be expressed quite naturally in our model. The 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.

V. IMPLEMENTATION

The system was developed into PostgreSQL [9] (written in *C*), as follows. When the user writes a query, PostgreSQL generates a data structure representing its corresponding relational algebra expression. After this data structure is generated, our Mining Extension is called (see Figure 1). Here, we process the relational algebra structure, extract the constraints, trigger the data mining algorithms and materialize the results in the virtual mining views. Just after the materialization, the workflow of the DBMS continues and the query is executed as if the patterns or models were there all the time.

The system is currently linked to algorithms for association rule discovery and exhaustive decision tree learning [5]. All the constraints listed in Section IV-B and represented as attributes of the mining views (size, accuracy, support, confidence) can be extracted and efficiently exploited by the integrated data mining algorithms. Some types of constraints, however, are currently not extracted by our implementation (for instance, the constraint "maximal accuracy" in query (B)); others are extracted but cannot be exploited by the mining algorithms. This may affect the efficiency of the system but not its correctness: the remaining constraints can be used to filter the results afterwards.

A. Experiments

We now present a set of experiments, which were conducted for the UCI dataset *ZOO* [10], with 101 examples, for the SQL queries showed in Figure 5. For queries (B), (C), and (D), the target attribute was the attribute "Class", the tests "Outlook=Sunny" and "Wind=Weak" were replaced by "Hair=true" and "Feathers=false", respectively, and the test "Temp=?" was changed to "Feathers=?". Our platform was an AMD ATLON 3.2 GHz processor, with 1 GB of memory, using Linux.

Table I presents the total execution times (in seconds), the number of intermediate generated concepts, rules (when applicable), trees (when applicable), and the size of the output (in rows) for the example queries.

For query (A), the constraints "supp>=30", "conf>=80" and "sz<=4" were all exploited by the system. Observe that the number of rows in its output corresponds to the exact number of rules that were intermediately generated.

For query (B), the constraint "sz<=5" was also exploited. This was not the case of the constraint "max(accuracy)", however. Yet, this query was correctly computed. The output consisted of 9 trees with 31 concepts, in total.

Regarding queries (C) and (D), the constraints "sz<=5" and "acc>=70" were both exploited. The constraints on the concepts that describe the decision trees are examples of constraints that were extracted by the system, but not exploited by the data mining algorithms. The results were nevertheless correctly computed.

As can be seen in table I, the execution times of the queries are rather low, which shows the usefulness and elegance of the proposed approach. The execution times consist mainly of the time spent by the data mining algorithms plus the time for materializing the results.

VI. DEMONSTRATION

The demonstration will focus on showing how the system works on different datasets using a set of constraints, such as those presented in Figure 5. Every time a data table T is created in the system, all virtual mining views associated with T are automatically created and the user can immediately query for itemsets, association rules or decision trees over T.

ACKNOWLEDGMENT

[redacted]

REFERENCES

[1] T. Imielinski and H. Mannila, "A database perspective on knowledge discovery," *Communications of the ACM*, vol. 39, no. 11, pp. 58–64, 1996.

[2] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, 2005.

[3] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler, "Yale: Rapid prototyping for complex data mining tasks," in *Proc. 12th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*. ACM, 2006, pp. 935–940.

[4] T. Calders, B. Goethals, and A. B. Prado, "Integrating pattern mining in relational databases," in *Proc. 10th European Conf. on Principles and Practice of Knowledge Discovery in Databases, PKDD*. Springer, 2006.

[5] E. Fromont, H. Blockeel, and J. Struyf, "Integrating decision tree learning into inductive databases," in *Knowledge Discovery in Inductive Databases (KDID), 5th International Workshop, Revised Selected and Invited Papers*, S. Dzeroski and J. Struyf, Eds., 2007.

[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 1994, pp. 487–499.

[7] H. Blockeel, T. Calders, E. Fromont, B. Goethals, and A. Prado, "Mining views: Database views for data mining," in *ECML/PKDD-2007 International Workshop on Constraint-Based Mining and Learning (CMILE)*, 2007.

[8] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[9] [Online]. Available: http://www.postgresql.org/

[10] D. N. A. Asuncion, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html