Decision support queries for the interpretation of data mining results

Bart Goethals, *Jan Van den Bussche*, *Koen Vanhoof*
Limburgs Universitair Centrum

Abstract

The interpretation of data mining results is an important step in the KDD process. This paper argues that many aspects of this post-processing phase can be conveniently viewed from the perspective of decision support queries. To realize this view, it suffices to use a structured representation of the data mining results, and to store these results together with the data themselves. The usefulness of the proposed approach is illustrated with an application from the Marketing domain.

Keywords: association rules, post-processing, decision support queries, filtering, marketing

Contact author: Jan Van den Bussche
LUC
Departement WNI
Universitaire Campus
B-3590 Diepenbeek, Belgium
+32-11-268226 (tel)
+32-11-268299 (fax)
`vdbuss@luc.ac.be` (email)

1 Introduction

The step following the data mining step in the KDD process consists of the interpretation of the data mining results [6]. This post-processing phase is non-trivial, since data mining results can be considerably large. We must thus provide the user with tools that allow him to infer from the results the information that really interests him.

Some authors even suggested terms such as "second-order data mining" or "rule mining" in this context. However, these terms are of course a little bit exaggerated. In the field of databases we know very well how to extract the information we are interested in from a given information base: this is what we know as querying.

Mining differs from querying in that, rather than extracting information *from* the base information, it produces meta-level information: information *about* the base information. Nevertheless, the basic idea we want to put forward in this paper is that unlike the mining step itself, many aspects of the post-processing step can already conveniently be handled by querying, using a general-purpose query language without any need for new syntactical constructs. This paper also fits in the subject of "data mining query languages," as we will explain later.

We develop our idea in the specific context of mining association rules [2]. Thereto, we propose a natural format under which the association rules, generated in the data mining step, can be stored in the database together with the data themselves. Storing meta-level data together with ordinary data is not unusual; indeed, the data dictionaries of most relational database systems consists of tables, holding meta data such as relation names, attribute names, etc., that can be accessed (not updated) like any other table in the database.

We will show that under the above view, all post-processing operations that we have found proposed in the literature can be performed by standard SQL queries. Put differently, our goal is to demonstrate that standard relational database systems, used in a perhaps slightly unconventional way, offer a powerful platform on top of which data mining post-processing tools can be implemented. Just for the same reasons all data processing applications are nowadays developed on top of a database system, we believe this approach is superior to incorporating post-processing features in KDD systems from scratch every time in an ad-hoc manner. Particular proposals which are naturally subsumed by our approach include the rule querying feature of the DataMine system [10], the mining conditions of Meo et al. [13], the rule templates of Klemettinen et al. [11], and the rule covers and groupings of Toivonen et al. [16].

Furthermore, we show that also specific post-processing needs of particular applications are covered by our approach, by presenting a concrete application in the Marketing domain. The application deals with customer satisfaction tests. As illustrations we will show how so-called "basic patterns of dissatisfaction" can be detected, and how customer satisfaction tests can be monitored.

As already mentioned, the queries produced by our approach are expressible in

SQL, although they are not especially simple. Most of them involve grouping, aggregate operators, and the comparison of nested subqueries: the typical characteristics of the class of queries known as *decision support queries*. We thus see that decision support queries, which usually are more associated with OLAP, also have a role to play in the KDD process. In general, non-standard query processing and optimization techniques are required to allow efficient execution of decision support queries on large databases [14, 5]. Much research has still to be done here, and will be done given that current interest in OLAP and decision support is tremendous.

However, the ideal platform for implementation of our approach is not relational at all, but is rather provided by OQL, the standard query language of object-oriented database systems (OODBs) [4]. Indeed, we will see that our model of storing the metadata, as well as our queries used to process them, are made much more transparent by the more powerful data modeling and query processing facilities offered by OODBs and OQL [1, 12]. We are convinced that OODB technology, which has matured in recent years, has a useful role to play in the KDD process and, for that matter, in decision support queries in general.

This paper is organized as follows. Section 2 describes our model of storing metadata. Section 3 considers various filtering operations, an important kind of post-processing. Section 4 presents the Marketing application. Section 5 concludes with a discussion of the notion of data mining query language.

2 Representation of mining results

As already mentioned in the Introduction, we focus in this paper on the mining of association rules [2].

Preliminaries Datasets are assumed to be stored in a relational database and are represented naturally as two-column tables. Each row in a data table is a pair of the form (t, i), where t is a *transaction identifier* (TID for short) and i is an *item*. Items can either be numbers which have some meaning known to the user, or can be references to a row (or rows) in another table (or tables) in the database that hold additional information on the item. For example, in market basket analysis, items are product codes, and there could be other tables in the database associating additional information to product codes such as product prices.

A TID t occurring in a data table D represents a set of items, denoted by $D(t)$, in the obvious way: $D(t) := \{i \mid (t, i) \in D\}$. As for items, there could be other tables in the database associating additional information to each transaction (other than its set of items), such as the date and time (in a market basket analysis application).

The mining of association rules is based on the notion of support of an itemset. The *support* of a set X of items is the percentage of TIDs in D that cover X. Here, a TID t is said to *cover* X simply if X is a subset of $D(t)$. An *association rule* has

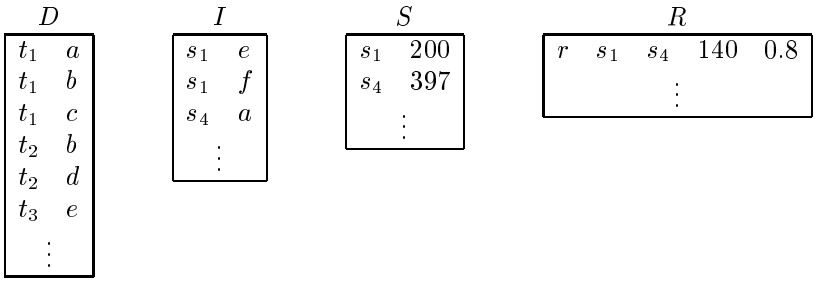

Figure 1: Simple abstract example of a data table D, an itemset table I, a support table S, and a rule table R.

the form $X_1 \Rightarrow X_2$, where X_1 and X_2 are itemsets. The *confidence* of this rule equals the support of $X_1 \cup X_2$ divided by the support of X_1. The *support* of this rule equals the support of $X_1 \cup X_2$. The mining step generates all association rules with support and confidence at least certain tresholds.

Representation of mining results The association rules, resulting from the mining of some data table, can be stored in the same database that contains the data tables, using a simple and natural format which we now describe.

1. *The itemset table.* All itemsets that occur in the association rules are stored in a two-column table. Thereto, we associate a unique identifier to each itemset. We then store in the table all pairs (s, i) where s is an itemset identifier and i is an item in the set identified by s.

2. *The support table.* The supports of the itemsets in the itemset table are stored in a separate two-column table holding all pairs (s, σ), where s is an itemset identifier and σ is the support of the set identified by s.

3. *The rule table.* The association rules are stored in a five-column table. For each rule $X_1 \Rightarrow X_2$ with support σ and confidence γ, there is a row $(r, s_1, s_2, \sigma, \gamma)$ in the table, where r is a unique identifier for the rule, s_1 is the identifier of X_1, and s_2 is the identifier of X_2.

A simple abstract example is given in Figure 1, which shows a data table D, an itemset table I, a support table S, and a rule table R. The data table contains, among others, TID t_1 representing the set of items $\{a, b, c\}$. The itemset table contains, among others, identifier s_1 of the itemset $\{e, f\}$. The support table shows, among others, that the support of $\{e, f\}$ is 200. The rule table shows that, among others, $\{e, f\} \Rightarrow \{a\}$ is an association rule with identifier r, support 140, and confidence 0.8.

Querying As a first illustration of how these tables can be queried, we show how an important auxiliary table for decision support and post-processing can be derived

```
select I1.sid, I2.sid
from Itemsets I1, Itemsets I2
where not exists
  (select I3.item
   from Itemsets I3
   where I3.sid=I1.sid and I3.item not in
     (select I4.item
      from Itemsets I4
      where I4.sid=I2.sid))
```

```
select s1, s2
from s1 in Itemsets, s2 in Itemsets
where s1.elements <= s2.elements
```

Figure 2: The subset query in SQL (left) and in OQL (right).

from the itemset table in SQL. Assume we have an itemset table called `Itemsets` with column names `sid` (for set identifier) and `item`. The query shown in Figure 2 (left) produces the table consisting of all pairs (s_1, s_2) of itemset identifiers such that the itemset identified by s_1 is a subset of the itemset identified by s_2.

Object-oriented representation If an object-oriented database system is available, we can exploit its more powerful data modeling and query processing capabilities [1, 12] and do better.

Basically we assume we have a class `Item` the instances of which are items, and a class `Transaction` the instances of which are transactions. Every `Transaction` object has (among others) a property `items` of type `set(Item)`. This means that if `t` denotes a `Transaction` object, `t.items` denotes a set of `Item` objects. A data table is then simply a set of `Transaction` objects.

To store itemsets and association rules, we define a class `Itemset` the instances of which are itemsets. Like `Transaction` objects, every `Itemset` object has (among others) a property `elements` of type `set(Item)`. Every `Itemset` object also has a property `support`. Furthermore, we define a class `Rule` the instances of which are association rules. Among others, `Rule` objects have properties `body` and `head` of type `Itemset`, and properties `support` and `confidence`. Itemset tables and rule tables now simply become sets of `Itemset` and `Rule` objects, respectively. There is no longer need for a separate support table since the support of an itemset is now directly modeled as a property of the corresponding object.

Figure 3 shows the same data shown in Figure 1 in object-oriented form.

Using OQL. To illustrate the advantage of using the object-oriented representation, let us reconsider the subset query of Figure 2 in this setting. In OQL, the standard object-oriented query language [4], the less-than (`<=`) comparison operator, when applied to set values, is automatically interpreted as subset-of. As a consequence, the subset query becomes much simpler when written in OQL, as shown in Figure 2 (right), and thus also easier to optimize and process by the system.

For simplicity of presentation, in the sequel we will use the object-oriented repre-

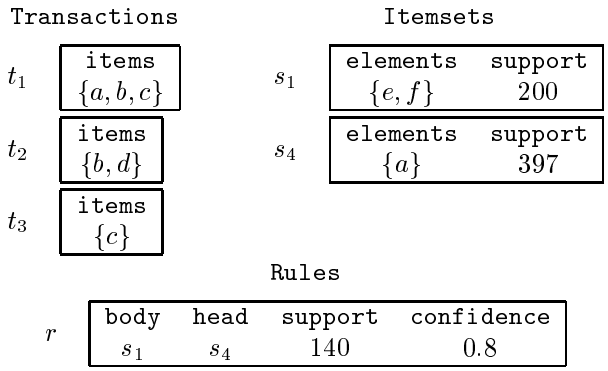

Figure 3: Object-oriented representation.

sentation of mining results, and we will use OQL as our query language. However, everything we will do can also be done under the relational representation using SQL.

3 Filtering of association rules

Let us now illustrate how filtering operations on sets of association rules—an important kind of post-processing operations—are readily expressed as standard queries in OQL (or SQL), under the natural representation of rules given in the previous section.

Templates A quite general mechanism for specifying typical filterings of association rules was introduced by Klemettinen et al. [11] in the form of "templates". Templates are boolean combinations of two basic kinds of conditions: *(i)* an upper or lower bound on the cardinality of the body or head of a rule; *(ii)* a given item that must, or must not, appear in the body or the head.

Templates are expressible as straightforward selection queries on the rule table. For example, to select those rules with a body of at least 3 items, among which the item a, and with a head not containing the item b nor c, we write:

```
select r
from r in Rules
where count(r.body)>=3 and a in r.body
and b not in r.body and c not in r.body
```

Mining conditions Meo et al. [13] suggested the possibility of putting qualitative conditions on the items in the head or body of a rule, which they called "mining conditions". Again this amounts to a selection query on the rule table. For example, to select those rules (in a market basket analysis application) having only expensive items (costing more than a certain price p) in the head, we write:

```
select r
from r in Rules
where forall i in r.head : i.price > p
```

Here the object-oriented representation we are using allows us to naturally assume that `Item` objects have a property `price`. In the relational representation, this would require an extra join with a `Prices` table. (The `forall` quantifier of OQL needs to be simulated in SQL using a `not exists` construction.)

Local versus global filtering Both templates and mining conditions are "local" filtering conditions, in the sense that whether or not a particular rule is filtered out depends only on that rule itself, not on how it compares to other rules. However, using a general-purpose query language to express filterings, it becomes possible to express also "global" rule filterings based on conditions that compare rules with each other, or even relate the generated rules back to the data.

As an illustration, consider the following filtering condition introduced by Toivonen et al. [16] under the term "structural rule covering:" *a rule should only be retained if there is no more general rule with a higher confidence.* Here, we call a rule $X_1' \Rightarrow X_2'$ *more general* than a rule $X_1 \Rightarrow X_2$ if $X_1' \subseteq X_1$ and $X_2 \subseteq X_2'$. So, a more general rule requires less items in the body, and still infers more items in the head.

It is quite reasonable to discard a rule if there is a more general rule with at least the same confidence. The set of rules that can be thus discarded is easily defined by the following OQL query:

```
select r
from r in Rules
where exists r1 in Rules : r1.body <= r.body and
r.head <= r1.head and r1.confidence >= r.confidence
```

We have experimented with this filtering operation on rules generated for real market basket data from a Belgian supermarket chain. To our suprise, it turned out that no rules at all were discarded in this way! Indeed, many rules had more general ones, but these invariably had less confidence. (Of course this is an empirical observation; in theory it is very well possible that a more general rule has more confidence.) However, when we slightly relaxed the restriction on the confidence of the more general rule, so that it may be less as long as it does not deviate drastically (more than 10%), we obtained a filter discarding almost 80% of the rules in our case.

Rule covers Another global filtering operation, also proposed by Toivonen et al., is to discard a rule r if there is another rule r', with at least the same confidence, such that the support set of r' is a superset of the support set of r. Here, the *support set* of a rule $X_1 \Rightarrow X_2$ is simply the set of transactions in the data table that cover $X_1 \cup X_2$. Again the set of rules that can be thus discarded is readily defined by an OQL query:

```
select r
from r in Rules
where exists r1 in Rules : r1.confidence >= r.confidence and
forall t in Transactions : covers(t,r1) or not covers(t,r)
```

Here, *covers*(t, r) is an abbreviation for `(r.body <= t.items and r.head <= t.items)`.

Note that this query actually involves both the rule table and the data table. As such it is not a purely "structural" filter, going beyond strict post-processing of generated rules. The possibility of post-processing rules by relating them back to the data was also hinted at by Imielinski et al. [10]. Expressing such operations does not pose a problem in our approach, since we store the rule table in the same database as the data tables.

Performance considerations Performance-wise, global filtering operations, such as the last two OQL queries shown, are rather heavy. A powerful database server, optimized for decision support queries, is needed to get fast response times. This is the price we pay (literally!) for almost unlimited flexibility in post processing.

4 Application: customer satisfaction

In the previous section, we considered filtering operations that are generally relevant. In this section, we consider post-processing operations specific to an application in the Marketing domain, namely, the monitoring of customer satisfaction tests.

In this application, we analyze the answers to a customer questionnaire. The questionnaire contains a number of specific questions, plus one question asking for his global assessment. To simplify our discussion, we assume only two possible answers to this last question: "dissatisfied" and "satisfied". So we define an item in this application as an answer to one of the questions; in particular, the two global answers *dissat* and *sat* are items. We then identify each customer by a transaction, the items of which are precisely the answers given by that customer.

Basic patterns and excitement patterns Studies on customer satisfaction [3] suggest that services may be grouped in two categories: those fulfilling minimum requirements, and those providing a genuine added value to the customer's experience. To get information on customer's expectations, we thus want to mine for *basic patterns*: patterns that cause dissatisfaction when not present but do not cause satisfaction when present; and for *excitement patterns*: patterns that cause satisfaction when present but do not cause dissatisfaction when not present. The treatment of excitement patterns is analogous to that of basic patterns. So we concentrate on the latter.

We can formalize a *pattern* as an itemset not containing the items *dissat* or *sat*. So a pattern is a set of answers to the other questions, used to try to predict the

global assessment. A *basic pattern* then can be formalized as a pattern X satisfying the following properties: *(i)* the rule $X \Rightarrow \{sat\}$ has low confidence; and *(ii)* the support of the single-item set $\{dissat\}$ does not change significantly if we ignore all transactions in the data table that cover X.

However, the above formalization does not give us a clue as to which patterns to test. It is practically impossible to generate and try them all. The solution is to look instead for basic patterns of *dis*satisfaction. So, these are the patterns that cause dissatisfaction when present, but do not cause satisfaction when not present. Formally, we look for the patterns X satisfying the following properties: *(i)* the rule $X \Rightarrow \{dissat\}$ has high confidence; and *(ii)* the support of $\{sat\}$ does not change significantly if we ignore all transactions in the data table that cover X.

Property *(i)* now allows us to test only those patterns that have sufficient support if we consider only the dissatisfied customers. This means we perform frequent itemset mining on the subset of the data table defined by the query `select t from t in Transactions where` *dissat* `in t.items`. We store the itemsets thus generated in an itemset table called `DissatPatterns`.

Finally, we execute the query shown in Figure 4 which gives us the complements of the basic patterns, as desired. In our experiments we processed this query on the customer satisfaction tests of a Belgian bank. The set `DissatPatterns` consisted of 1195 patterns. From these, 25 were selected by the query with value 85 for parameter h. An example of the outcome is that the presence of bank clerks who are impatient, impersonal, and unclear in their explanations, will generally cause global dissatisfaction of the customer.

The query of Figure 4 is another example of a filtering operation that relates the results of mining back to the data. Performance-wise the query is quite heavy, and will have to rely on good optimization and processing techniques. Obviously there is a trade-off here between what we want to compute in the mining step and what in the post-processing step. Indeed, we can alternatively customize the mining algorithm and compute the needed confidences and supports there. Then the query becomes a trivial selection query.

Monitoring customer satisfaction Other interesting post-processing operations one can perform concern monitoring: comparing the results of the current customer survey and a previous one (e.g., excitement patterns may well evolve into basic patterns after some time!) Monitoring is again easy to perform by standard querying operations; due to space limitations we omit all details.

5 A perspective on data mining query languages

The notion of *data mining query language* has appeared in a number of works [8, 10, 13, 9]. The idea is to have a language in which the complete data mining step can

```
select p
from p in DissatPatterns
where count(select t
            from t in Transactions
            where t.items >= p and dissat in t.items)
      / count(select t
              from t in Transactions
              where t.items >= p) >= h
 and  count(select t
            from t in Transactions
            where not (t.items >= p) and sat in t.items)
      / count(select t
              from t in Transactions
              where not (t.items >= p)) ≃ σ
```

Figure 4: Finding the complements of basic patterns. Here, h is some treshold value, σ is the support of $\{sat\}$, and $x \simeq y$ means that x and y are not significantly different (e.g., $x \simeq y \Leftrightarrow |x - y| \leq 2$).

be expressed. This includes the specification of many aspects of that step, including: *(i)* how the actual data table (on which mining has to be performed) is obtained from the tables in the database; *(ii)* which attributes determine the TID, and which determine the items; *(iii)* minimum support and confidence tresholds; *(iv)* filtering conditions on the generated rules.

Aspect *(i)* is nothing but a query. We have shown in this paper that, under a natural representation of the generated rules, aspect *(iv)* is also nothing but a query. Aspects *(ii)* and *(iii)* are rather trivial.

It would be very nice if we could design a powerful query language in which not only the above four aspects, but also the mining algorithm itself can be expressed as "just another query". Such an "ultimate" data mining query language could look very much like a complex-object query language such as OQL, extended with iteration capabilities so that the iterative nature of rule generation in association rule mining algorithms such as Apriori [2] can be captured.[1] It would then be possible to express mining and filtering together as one global query operation, which opens the perspective of optimization techniques that incorporate the filtering of rules directly within the generation of these rules (akin to the basic optimization technique of pushing selections within joins in relational query processing [17]). Optimizing and processing the query of Figure 4 is an interesting example challenge.

Of course, to make this dream come true, the crucial challenge is to investigate whether there are query processing and optimization techniques for complex-object query languages with iteration, so that the complexity of mining executed as a query is not much higher than that of mining executed by a special-purpose program.

[1]For an exposition on query languages with iteration or recursion capabilities, see Abiteboul, Hull, and Vianu's book [1].

Acknowledgment

The second author would like to thank Dirk Van Gucht for a number of clarifying discussions on query languages for decision support and data mining.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In Fayyad et al. [7], pages 307–328.

[3] D.R. Brandt. How service marketers can identify value-enhancing service elements. *Journal of Services Marketing*, 3, 1988.

[4] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1994.

[5] S. Chaudhuri and K. Shim. Including group-by in query optimization. In J.B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proceedings 20th International Conference on Very Large Data Bases*, pages 354–366. Morgan Kaufmann, 1994.

[6] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In Fayyad et al. [7], pages 1–34.

[7] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.

[8] J. Han et al. DBMiner: A system for mining knowledge in large relational databases. In Simoudis et al. [15], pages 250–255.

[9] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[10] T. Imielinski, A. Virmani, and A. Abdulghani. *DataMine*: Application programming interface and query language for database mining. In Simoudis et al. [15], pages 256–261.

[11] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In N.R. Adam, B.K. Bhargava, and Y. Yesha, editors, *Proceedings 3rd International Conference on Information and Knowledge Management*, pages 401–407. ACM Press, 1994.

[12] G. Lausen and G. Vossen. *Models and Languages of Object-Oriented Databases.* Addison-Wesley, 1997.

[13] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *Proceedings 22nd International Conference on Very Large Data Bases*, pages 122–133. Morgan Kaufmann, 1996.

[14] S. Rao, A. Badia, and D. Van Gucht. Providing better support for a class of decision support queries. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, volume 25:2 of *SIGMOD Record*, pages 205–216. ACM Press, 1996.

[15] E. Simoudis, J. Han, and U. Fayyad, editors. *Proceedings 2nd International Conference on Knowledge Discovery & Data Mining.* AAAI Press, 1996.

[16] H. Toivonen et al. Pruning and grouping discovered association rules. In *MLnet Workshop on Statistics, Machine Learning, and Discovery in Databases*, Herakлion, Crete, Greece, April 1995.

[17] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.