A priori versus a posteriori filtering of association rules

(extended abstract)

Bart Goethals
Limburgs Universitair Centrum
bart.goethals@luc.ac.be

Jan Van den Bussche
Limburgs Universitair Centrum
jan.vandenbussche@luc.ac.be

1 Introduction

The concept of *inductive database*, proposed by Mannila [8, 11], is a beautiful formalization of the interactive mining process. In the concrete setting of association rule mining [1], an inductive database provides virtual tables containing virtually all itemsets and rules over the data. The user does not care how these inductive tables are implemented; for him, mining is nothing but querying these tables.

For example, in a market basket application, suppose we want all rules (with a certain confidence and support as usual) having 'banana' in the head but 'corn flakes' not in the body. This would be expressed by the following data mining query:

select *r.head*, *r.body*, *r.confidence*, *r.support*
from *Rules r*
where *r.support* ≥ 10 **and** *r.confidence* ≥ 65
and 'banana' **in** (**select** *itemid*
from *Sets*
where *setid* = *r.head*)
and 'corn flakes' **not in** (**select** *itemid*
from *Sets*
where *setid* = *r.body*)

Note that such queries might even combine the *Sets* and *Rules* tables with the given data tables, and that mining is an essentially interactive process, where a user repeatedly poses new queries based on what he found in the answers of his previous queries.

In our opinion, the idea of inductive database indicates the ultimate goal of how a transparent "data mining query language" [5, 4, 6, 7, 12] should look like. The transparency lies in that the user never issues explicit mining commands himself; the system mines whatever and whenever necessary. Clearly the implementation of this vision presents a great challenge. In this paper, we investigate and compare two rather extreme approaches, the *a priori approach* and the *a posteriori approach*, towards the above challenge.

The a priori approach consists of answering every individual data mining query by running an adaptation of the mining algorithm in which the conditions on the rules to be generated (as specified in the query) are directly incorporated. For example, to answer the above example query, one would try to generate only the rules with 'banana' in the head and 'corn flakes' not in the body, without generating irrelevant rules or itemsets. Such adaptations of the Apriori algorithm have already been considered in the literature, but the problem of how to do this for a wide variety of conditions is definitely not yet completely solved. In this paper we will offer a further contribution in this direction.

The a posteriori approach begins by filling up the *Sets* and *Rules* tables as densely as possible, by performing one major, global mining operation where the minimal support and confidence parameters are set as low as one would possibly need. After this relatively expensive operation, the actual data mining queries issued by the user then amount to standard, basic queries on the materialized tables.

2 Filtered mining of association rules

In this section, we introduce a class of filters, i.e., conditions on rules, and show a way to integrate these filters tightly in the mining algorithm.

The filters we will consider in this paper are a special case of the rule templates introduced by

Klemettinen et al. [9]. Concretely, we define a filter as a conjunction of *basic conditions*, where a basic condition specifies that some special item must or must not occur in the body, the head, or anywhere in the rule. An example of a filter is: 'a in body and b in head and c not in rule'.

The question of how such filters (and other kinds of conditions on rules) can be exploited in the mining algorithm has already been considered by Srikant, Vu, and Agrawal [16] and Lakshmanan, Ng, et al. [10, 13]. The latter work uses the concept of "member generating function" as an aid to restrict the generation of itemsets to only those satisfying the filter. The former work is approximate, but can deal with disjunctions as well. If a filter exists of only one disjunct however, a lot of problems mentioned in their work disappear, which makes it possible to simplify and optimize the algorithms.

We will present a filtered mining algorithm that (on conjunctions) is conceptually clearer and also more efficient than earlier algorithms by adapting a combination of the Reorder and Direct algorithms of Srikant et al. The filtering achieved is non-redundant, in the sense that it never generates an itemset that could give rise to a rule that does not satisfy the filter, and it avoids the detour via member generating functions. A specific feature of our algorithm is that we do not index the generated itemsets by hashing, but by a trie, the standard data structure for indexing collections of strings (and hence also ordered sets). The use of a trie allows a very direct and natural incorporation of filters, and also offers various other advantages [2, 3].

Let $b_1, \ldots, b_\ell$ be the items that must be in the body by the filter; $b'_1, \ldots, b'_{\ell'}$ those that must not; $h_1, \ldots, h_m$ those that must be in the head; $h'_1, \ldots, h'_{m'}$ those that must not; $r_1, \ldots, r_n$ those that must be anywhere in the rule; and $r'_1, \ldots, r'_{n'}$ those that must not.

Recall that an association rule $X \Rightarrow Y$ is only generated if $X \cup Y$ is a frequent set. Hence, we only have to generate those frequent sets that contain every b_i, h_i and r_i, plus some of the subsets of these frequent sets. This can be done as follows:

1. Start an initial trie with the linear chain $b_1, \ldots, b_\ell, h_1, \ldots, h_m, \ldots, r_1, \ldots, r_n$, adding at the bottom level all other items as leafs, except that we ignore the "negative" items r'_i. The leafs represent all candidate itemsets of size $k+1$, where $k = \ell + m + n$. We thus start with a lead of k in comparison with standard, non-filtered mining. From here on, we perform the standard iteration: count frequencies; delete infrequent itemsets; generate candidate sets of size $k+2$; prune; and repeat. Note that a slight downside of filtered mining is that, while in non-filtered mining we can prune by testing if *all* subsets of a candidate set are frequent, here we can only test all subsets containing every b_i, h_i and r_i, simply because the other subsets have not been generated and hence their frequencies are unknown.

2. We now have all frequent sets containing every b_i, h_i and r_i. In order to generate rules, we also need those subsets of these sets that can serve as bodies. These subsets must contain every b_i, and none of the b'_i or h_i (the latter because bodies and heads of rules are disjoint). Furthermore, we need those subsets that can serve as heads; these must contain every h_i, and none of the h'_i or b_i. It is very easy to add all the needed subsets to the trie and determine their frequencies in one additional pass.

3. We finally generate the desired association rules from the appropriate sets generated in steps 1 and 2, in accordance with the filter conditions. The prunings that can be performed in this step [15] can again be very easily implemented through the built-up trie.

We performed some modest experiments which clearly confirm the speedups achieved by filtered mining, as expected theoretically due to the non-redundancy of our algorithm. Note that a filter consisting of a single condition only has the least effect; hence, the speedups achieved for such filters serve as a lower bound. We have run the six possibilities for a one-condition filter three times: once for an item with high frequency, once for an item with average frequency, and once for an item with low frequency. Our results, depicted in Figure 1, show that filtering has more effect on low-frequency items (which is quite intuitive). Notice that the speedup gained with purely negative

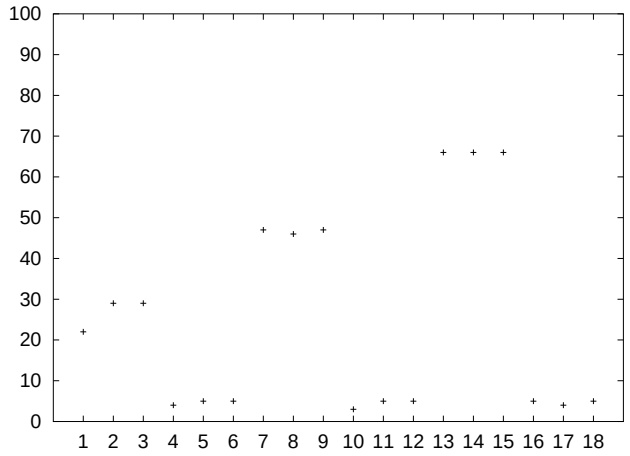

Figure 1: Speedups (in percentages) gained by filtered mining, for one-condition filters of the six forms 1: 'i in head'; 2: 'i in body'; 3: 'i in rule'; 4: 'i not in head'; 5: 'i not in body'; and 6: 'i not in rule'. Numbers 1–6 are for an i with high frequency, numbers 7–12 for average frequency, and numbers 13–18 for low frequency.

conditions (such as '187 not in body') is smaller, because such a condition can only be exploited in steps 2 and 3 above, not in the most costly step 1.

3 A priori versus a posteriori

In the previous section, we have seen a way to integrate filter conditions tightly into the mining of association rules. We call this *a priori filtering*. At the other end of the spectrum we have *a posteriori filtering*, where we perform standard, non-filtered mining, only after the completion of which we filter the result using a standard query.

A priori filtering has the following two obvious advantages over a posteriori filtering:

1. Answering one single data mining query using a priori filtering is much more efficient than answering it using a posteriori filtering.

2. It is well known that, by setting parameters such as minimal support too low, or by the nature of the data, association rule mining can be infeasible simply because of a combinatorial explosion involved in the generation of rules or frequent itemsets. Under such circumstances, of course, a posteriori filtering is infeasible as well; yet, a priori filtering can still be executed, if the filter conditions can be effectively exploited to reduce the number of itemsets and rules from the outset.

However, this is certainly not all that can be said. As already mentioned in the Introduction, data mining query language environments must support an interactive, iterative mining process, where a user repeatedly issues new queries based on what he found in the answers of his previous queries. Now consider a situation where the second advantage above does not apply, i.e., minimal support requirements and data set particulars are favorable enough so that a posteriori filtering is not infeasible to begin with. Then the global, non-filtered mining operation, on the result of which the filtering will be performed by a posteriori filtering, *can be executed once and its result materialized for the remainder of the data mining session* (or part of it).

In this case, if the session consists of, say, 20 data mining queries, these 20 queries amount to standard retrieval queries on the materialized mining results. In contrast, answering every single of the 20 queries by an a priori filter will involve at least 20, and often many more, passes over the data, as each of these a priori filters involves a separate mining operation. The naively conceived obvious advantages of a priori filtering over a posteriori filtering have suddenly become much less clear now.

We can analyze the situation easily as follows. Consider a session in which the user issues a total of m data mining queries over a database of size n. Suppose that the total number of association rules (given a minimal support requirement) over these data equals r. Let t be the time required to generate all these rules. Moreover, it is not unreasonable to estimate that in a posteriori filtering, each filter executes in time proportional to r, and that in a priori filtering, each a priori filter executes in time proportional to n. Then the total time spent by the a posteriori approach is $t + m \cdot r$, while in the a priori approach this is $m \cdot n$. Hence, if $n > r$, then *the a priori total time is guaranteed to grow beyond the a posteriori total time;* indeed, this happens exactly at the cut-off point of $m = \lceil t/(n-r) \rceil$ queries.

We have modestly experimented with a realistic

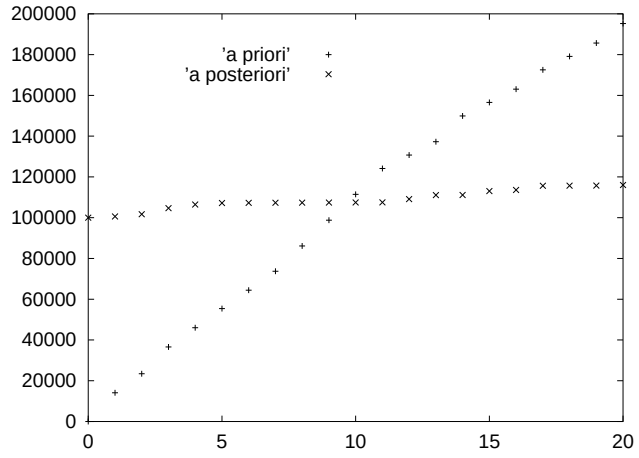

Figure 2: On the x-axis, the number of queries. On the y-axis, the total time in milliseconds elapsed during processing of the last x queries.

session of 20 data mining queries over real market basket data (from a Belgian chain of automated 24-hour shops) containing 300 000 transactions on 300 items. The cut-off point where the a priori approach loses against the a posteriori approach is reached after 9 queries. The evolution of the session in time, using a priori, and using a posteriori, is shown in Figure 2.

4 Challenges towards an implementation of inductive databases

One striking feature of the results of Figure 2 is how slowly the a posteriori curve grows. This means that, once the global mining operation has been performed (which took 100 seconds in our case, hence the curve starts at $y = 100\,000$), the times needed to answer the individual queries on the materialized tables is extremely small. We expressed these queries in SQL in the obvious way (as the example from the Introduction) and used SQL Server to answer them.

If the necessary indices are present on the *Sets* and *Rules* tables, queries of this style are indeed very quickly answerable by relational query processors. With queries that go beyond the simple template filters considered in this paper, such as queries that compare rules with each other, or queries that relate rules back to the data, the situation may of course be different. Much further work is needed in this direction.

More generally, a priori and a posteriori filtering are only two extremes in possible ways to implement an interactive data mining query language environment. The perfect way lies somewhere in the middle; we envisage it to work along the following lines.

Initially, when the user issues his first query, nothing has been mined yet (the *Sets* and *Rules* tables are empty), and thus the most efficient way to answer this particular first query is to use a priori filtering. The results of the mining involved in this first step are saved in the *Sets* and *Rules* tables.

Now to answer further queries, the system should be able to decide to what extent the rules needed to answer a query are already present, and to what extent new rules must be mined for. The filter condition must accordingly be factorized into one part that can simply be performed by querying the already materialized sets and rules, and another part that will be integrated into the mining operation using a priori mining. Again, many great challenges for further research in this direction remain.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. MIT Press, 1996.

[2] A. Amir, R. Feldman, and R. Kashi. A new and versatile method for association generation. *Information Systems*, 2:333–347, 1997.

[3] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, volume 26:2 of *SIGMOD Record*, pages 255–264. ACM Press, 1997.

[4] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. Presented at SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery.

[5] J. Han, Y. Fu, W. Wang, et al. DBMiner: A system for mining knowledge in large relational databases. In Simoudis et al. [14], pages 250–255.

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

[7] T. Imielinski, A. Virmani, and A. Abdulghani. *DataMine*: Application programming interface and query language for database mining. In Simoudis et al. [14], pages 256–261.

[8] M. Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.

[9] M. Klemettinen et al. 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.

[10] L.V.S. Lakshmanan, R.T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proceedings 1999 ACM SIGMOD International Conference*. To appear.

[11] H. Mannila. Inductive databases and condensed representations for data mining. In Jan Maluszynski, editor, *Logic Programming, Proceedings of the 1997 International Symposium*, pages 21–30. MIT Press, 1997.

[12] 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.

[13] R.T. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In L.M. Haas and A. Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27:2 of *SIGMOD Record*, pages 13–24. ACM Press, 1998.

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

[15] R. Srikant and R. Agrawal. Mining generalized association rules. In U. Dayal, P.M.D. Gray, and S. Nishio, editors, *Proceedings 21th International Conference on Very Large Data Bases*, pages 407–419. Morgan Kaufmann, 1995.

[16] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings 3rd International Conference on Knowledge Discovery & Data Mining*, pages 66–73. AAAI Press, 1997.