Fast Relational Data Mining

Query Optimization for Improving the Efficiency of Relational Data Mining Systems

Jan Struyf

Abstract—**Data mining is the process of building predictive or descriptive models based on a large data set, often stored in a relational database. Propositional data mining systems require that the data is converted into one single table. Relational data mining systems, on the other hand, can build models directly from the relational database. While building a model, relational data mining systems execute a huge number of queries on the database and this consumes much CPU time. In our work, we propose a number of query optimization techniques that speed up query execution. Relational data mining systems generate queries and access the data in a structured way. Our optimizations exploit this structure as much as possible.**

Keywords— **Relational Data Mining, Machine Learning, Efficient Algorithms, Query Optimization**

I. Introduction

DATA mining is the process of building predictive or descriptive models based on large data sets. Many types of data sources can be used for data mining. The data can be stored in a relational database (RDB), it can be a collection of HTML or XML documents, a set of DNA sequences or a number of molecule descriptions.

In predictive data mining, the goal is to build a model that maps an instance to a valid prediction, i.e. a class in case of a classification task or a real value in case of a regression task. For example, a predictive model could predict whether or not a given customer will buy a certain product or it could predict the mutagenicity of a molecule. Rule sets, decision trees, instance based representations, neural networks and support vector machines are frequently used predictive models.

Descriptive data mining algorithms are designed to discover interesting knowledge from the data, such as frequently occurring patterns or clusters of similar instances. Basket analysis is a typical descriptive data mining application where one is interested in sets of items that are frequently bought together, for example, by the customers of a supermarket.

II. Relational Data Mining

Data mining algorithms can be classified in several categories depending on the way the input data is represented. Many systems are designed to work on data stored in a single table. We call such systems propositional systems because their data representation is equivalent to propositional logic. In our work we focus on a second class of data mining systems: relational data mining systems. Relational data mining systems can learn directly from data stored in a RDB. The difference between propositional and relational data mining systems is depicted in Figure 1.

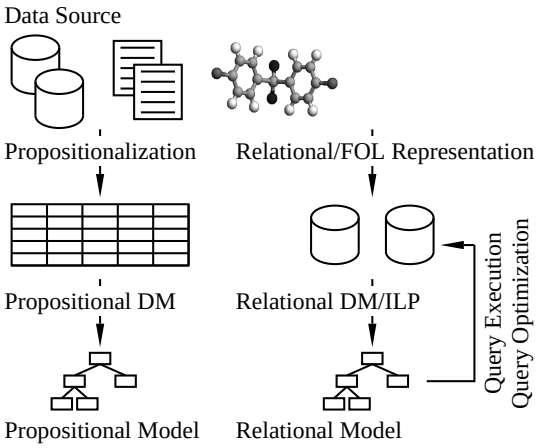

Fig. 1. Propositional and Relational Data Mining.

The main drawback of propositional systems is that the data has to be converted into a single table, where each row describes a specific instance with a fixed set of features or attributes. This conversion process, called propositionalization, is difficult, may require non-trivial human interaction and results in many cases in loss of information. Relational data mining systems do not require this complex conversion step because they can learn from the data in its original representation. Another benefit of relational data mining is that the output can also be formulated in terms of the original representation, which makes interpretation of the induced models easier for the domain expert.

Inductive Logic Programming (ILP) systems are data mining systems that can learn from data represented as a logic program. Logic programs are a subset of first order logic (FOL). Because FOL is a very expressive formalism, the representation used by many data sources (including RDBs) can be considered a special case. An advantage of ILP systems is that they can incorporate expert knowledge formulated as a FOL theory.

III. Query Optimization

Relational frequent pattern mining is a generalization of basket analysis. A relational pattern (or query) can be represented in FOL as a conjunction of first order literals. Consider the following example query (we represent variables with capital letters).

person(X) $\wedge$ buys(X, pizza) $\wedge$ friend(X, Y) $\wedge$
buys(Y, wine).

Jan Struyf is a PhD. student at the Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium, *jan.struyf@cs.kuleuven.ac.be*.

The frequency of this query is equal to the proportion of persons X in the RDB that buy pizza and have a friend Y who buys wine. If this frequency is above a given threshold, the query is called a frequent query.

In order to build a model from a RDB, an ILP system has to execute a huge number of queries on it. For example, for the relational pattern mining application, the system will execute a set of candidate queries on de data, compute the frequency of each candidate and output those that are found frequent. Because one query may involve several relations, query execution is much more complex than in the propositional case and consumes much CPU time. As a consequence, efficient query execution is essential for ILP systems.

One approach for building ILP systems is to connect them to an existing relational database management system (RDBMS). The advantage is that queries will be executed efficiently because such systems typically implement a number of query optimization techniques. However, we argue that it is better not to do this. Instead one should design a special purpose RDBMS. This special purpose RDBMS does not need to include all components of a regular RDBMS. For instance, it does not need to provide transactions and locking. These are not necessary for ILP and only cause extra overhead. The main advantage of using a special purpose RDBMS is that it can include optimizations that are specific for ILP systems. Such optimizations are possible because ILP systems generate queries and access data in a structured way. In our work we design and evaluate such specific optimizations. A possible optimization is to store all records relevant for a certain instance together. Because ILP systems access the database one instance at a time, these sets of records can be loaded one by one and cached easily in main memory.

In [1], we propose a number of query transformations that are specific to ILP. A query transformation transforms a query into a different form that can be executed faster. Our transformations exploit the fact that ILP systems for a given query and instance only need to know if the query succeeds or not. The entire answer set does not have to be computed. One of the transformations inserts special control constructs in the query informing the execution engine that only one solution is necessary for a certain subquery. Such a construct can be used if the query consists of several independent components. A second transformation exploits the fact that queries are generated by the ILP system by extending previously generated queries with a number of literals. Independent components for which the system knows that they succeed (from previous execution) can be removed from the query. A third optimization removes redundant literals from a query.

In [2] we propose a query transformation that reorders the literals of a query. Queries can be executed faster if selective literals are placed first. For example, if a certain literal only succeeds for 1% of the instances, then the remaining literals do not need to be executed for 99% of the data if the selective literal is executed first.

ILP systems generate queries in a structured way. A new query is generated by extending a preceding one with a number of literals. This implies that many queries share common prefixes. By converting the set of queries to a tree, computations for these common prefixes can be shared. In our research group, a special execution mechanism for these tree structures called query-pack execution has been proposed. In more recent work we have exploited the tree structure in an efficient frequent pattern mining system. Such a system must check each new candidate query it generates for equivalence with the preceding queries. We show in [3] that this equivalence check can also be made more efficient by exploiting the tree structure.

Finally, we are working on an execution mechanism that allows one to combine query transformations with query-pack execution. A first version of this method is described in [4]. We hope that this new method will combine the advantages of query transformations and query-pack execution.

IV. Conclusions

Relational data mining systems are data mining systems that can build models directly from the information stored in a relational database. Inductive Logic Programming (ILP) is one approach to relational data mining. ILP systems spend most of their execution time on query execution. We propose a number of query optimizations for speeding up query execution that are specific for ILP. Such optimizations are possible because ILP systems generate queries and access data in a structured way.

Acknowledgments

Jan Struyf is a research assistant of the Fund for Scientific Research - Flanders (FWO). The author would like to thank his supervisors Hendrik Blockeel and Bart Demoen and also Jan Ramon and Gerda Janssens for the many fruitful discussions.

References

[1] V. Santos Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W. Van Laer, "Query transformations for improving the efficiency of ILP systems", *Journal of Machine Learning Research*, vol. 4, pp. 465–491, Aug. 2003.

[2] J. Struyf and H. Blockeel, "Query optimization in inductive logic programming by reordering literals", in *Proceedings of the 13th International Conference on Inductive Logic Programming*. 2003, Lecture Notes in Artificial Intelligence, Springer.

[3] J. Ramon and J. Struyf, "Efficient theta-subsumption of sets of patterns", in *Proceedings of Benelearn 2004, the Annual Belgian-Dutch Conference on Machine Learning*, 2003, Submitted.

[4] R. Tronçon, H. Vandecasteele, J. Struyf, B. Demoen, and G. Janssens, "Query optimization: Combining query packs and the once-transformation", in *Proceedings of the 13th Internal Conference on Inductive Logic Programming*, 2003, Short Presentations.