

Efficient Multi-Relational Data Mining (extended abstract)

Hendrik Blockeel and Jan Struyf

Katholieke Universiteit Leuven, Dept. of Computer Science,
 Celestijnenlaan 200A, B-3001 Leuven, Belgium
 {Hendrik.Blockeel, Jan.Struyf}@cs.kuleuven.ac.be

Extended abstract

Multi-relational data mining algorithms search a large hypothesis space in order to find a suitable model for a given data set. During this search, a huge number of queries has to be evaluated on the data set. Evaluating a query that uses and combines information from different relations is far more complex comparing to evaluating a test on a single table the way propositional data mining algorithms do. The main reason for this is that in the multi-relational context, when executing queries top-down, backtracking can occur over different relations. Because query evaluation is more complex, multi-relational data mining algorithms typically have high run times and it becomes interesting to do research for techniques that speed up model building. The goal of this text is to give an overview of two such techniques. The first one is query-pack evaluation [2] and the second one is parallel cross-validation [3], [5].

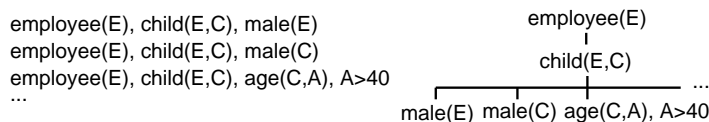


Fig. 1. A query-pack (Prolog notation).

A first order data mining algorithm usually selects queries by navigating through a lattice structure defined by the algorithm's refinement operator. First order decision tree induction systems (TILDE [1], S-Cart [4]) for example consider refinements of the query from the previous level of the tree when selecting a query for a new node. Because each refinement is obtained by extending the parent query with a few new literals, different refinements are highly similar (share literals). As a consequence, independent execution of all refinements may involve a lot of redundant computations. These redundancies can be removed by integrating the similar queries in a so-called query-pack [2]. A query-pack (Fig. 1) is a tree structure with literals or conjunctions of literals in the nodes. Each path from the root to some node represents a conjunctive query. Query-pack execution is more efficient comparing to independent execution because answer

substitutions from the trunk (branches) can be used at different places in the branches (sub-branches) and leaves. The theoretical speedup factor depends on the pack’s branching factor and on the computational complexity of the shared and non-shared part of the pack. If the branching factor is high and most of the complexity is concentrated in the shared part of the pack, then we can expect high speedups. This was validated experimentally by the authors of [2] by implementing query-pack execution in the ACE data mining tool¹. Running TILDE [1], the decision tree learner integrated in ACE, yields in the best case a speedup of 20 times.

Cross-validation can be used when insufficient data is available to reliably train on one subset of the data and validate the results on a disjoint subset. It consists of partitioning a data set D into n subsets D_i and then running a data mining algorithm n times, each time using a different training set $D - D_i$ and validating the results on D_i . The results on each D_i are averaged to provide a reliable estimate of the induced model’s performance on unseen cases. The obvious disadvantage of cross-validation is the computational overhead of running the data mining algorithm n times. However, for some data mining techniques, this overhead can be reduced significantly, again by removing redundancies. With query-pack execution, redundancies occurred when executing similar queries separately on a data set. Here the redundancies stem from running the same queries on similar data sets. Note that the training sets $T_i = D - D_i$ used in the different cross-validation folds are highly similar because each example from D is replicated $n - 1$ times in the different T_i . By building the n cross-validations models in parallel it is possible to remove these redundancies. The complexity of this parallel cross-validation algorithm is influenced by the number of refinements, the data access time, the time for updating statistics and the model stability (how similar are the models generated in different folds). If model stability and data access time is high and statistic update time low, then a speedup up till n times comparing to classical serial cross-validation can be achieved. Parallel cross-validation is explained in more detail in [3]. This work also provides experimental results based on an implementation of parallel cross-validation in TILDE. In the best case parallel cross-validation (10 folds) is about 9 times faster comparing to serial cross-validation.

It is also possible to combine query-pack execution and parallel cross-validation (we summarize the work from [5] - see Fig. 2). In this case we remove the two types of redundancies at once. Similar queries are evaluated efficiently using query-pack execution. Similar training sets are handled efficiently by parallel cross-validation. The obtained speedup will be maximal if query complexity is high (due to query-pack execution - this is the case for Mutagenesis) or if model stability is high (due to parallel cross-validation - this is the case for Simple Bongard). Note that high query complexity can imply low model stability. Combining query-pack execution and parallel cross-validation results in a robust algorithm that scores reasonable on a wide range of data sets.

¹ ACE [2] is available for academic purposes upon request.
<http://www.cs.kuleuven.ac.be/~dtai/ACE/>

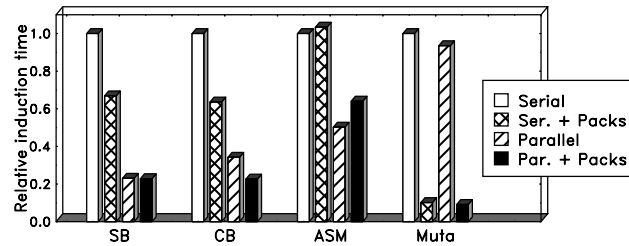


Fig. 2. Execution time relative to serial 10-fold cross-validation. Data sets are: Simple (1453 examples) and Complex (1521 examples) Bongard, Adaptive Systems Management (999 examples) and Mutagenesis (230 examples). Note that the current query-pack implementation does not handle the huge number of discretized numerical attributes of the ASM set efficiently.

We conclude with a final note on the applicability of the techniques discussed. Query-pack execution is very general and can be applied if many similar queries have to be evaluated on the same data set. Parallel cross-validation can be integrated in different data mining algorithms. It is shown in [3] and [5] how this can be done for decision tree and rule induction systems. The main limitation is that parallel cross-validation is not immediately applicable to algorithms that built continuous models, such as neural networks.

Acknowledgements

Hendrik Blockeel and Jan Struyf are a post-doctoral fellow, respectively research assistant, of the Fund for Scientific Research of Flanders.

References

1. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
2. H. Blockeel, B. Demoen, L. Dehaspe, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ILP. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference in Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 60–77, London, UK, July 2000. Springer.
3. Hendrik Blockeel and Jan Struyf. Efficient algorithms for decision tree cross-validation. In *Proceedings of ICML-2001 - Eighteenth International Conference on Machine Learning*, pages 11–18. Morgan Kaufmann, 2001.
4. Stefan Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 812–819, Cambridge/Menlo Park, 1996. AAAI Press/MIT Press.
5. Jan Struyf and Hendrik Blockeel. Efficient Cross-validation in ILP. In *Proceedings of ILP-2001 - Eleventh International Workshop on Inductive Logic Programming*, 2001. To appear.