Parallel Induction of Modular Classification Rules

Frederic Stahl[1], Max Bramer[1] and Mo Adda[1]

Abstract The Distributed Rule Induction (DRI) project at the University of Portsmouth is concerned with distributed data mining algorithms for automatically generating rules of all kinds. In this paper we present a system architecture and its implementation for inducing modular classification rules in parallel in a local area network using a distributed blackboard system. We present initial results of a prototype implementation based on the Prism algorithm.

1. Introduction

The field of Data Mining from large collections of data has experienced a considerable upsurge of commercial interest and research activity in recent years.

So far relatively little attention has been given to distributing or parallelising data mining algorithms, but as the size of datasets requiring analysis continues to increase it seems inevitable that this will become a major focus of attention. Most data mining algorithms make the implicit assumption that all the training data can be stored and processed in main memory. It is common practice to take a sample of stored data to form a training set for analysis, but as long ago as 1992, Catlett[1] pointed out the loss of predictive accuracy that can result from sampling very large training sets. The datasets considered very large at that time would be considered commonplace by today's standards. In scientific fields such as bioinformatics and cosmology it may be particularly important to process all the data available or risk overlooking valuable relationships in the data.

The Distributed Rule Induction (DRI) project at the University of Portsmouth is concerned with distributed data mining algorithms for automatically generating rules of all kinds. Most rule induction has been for the purpose of classification [2] and the most common approach to classification rule generation is via the intermediate form of a decision tree [2]. Although popular this method suffers from the problem of overfitting and it is generally considered desirable to post-

[1] School of Computing, University of Portsmouth, PO1 3HE, UK
{Frederic.Stahl; Max.Bramer; Mo.Adda}@port.ac.uk

prune the trees generated before converting them to rules and then to process the rules further to remove overfitted terms as far as possible [3]. This seems an unnecessarily indirect way to generate rules, especially when there are algorithms that will generate them directly, albeit ones that are less widely known.

The Prism algorithm was developed as an alternative to decision tree generation [4]. For continuous data the algorithm can be summarised as follows, assuming that there are n (>1) possible classes [5].

```
For each class i from 1 to n inclusive:
(a) working dataset W = initial Dataset;
    delete all records that match the rules that have
    been derived so far for class i;
(b) For each attribute A in W
  - sort the data according to A;
    - for each possible split value v of attribute A
      calculate the probability that the class is i
      for both subsets A < v and A ≥ v;
(c) Select the attribute that has the subset S with
    the overall highest probability;
(d) build a rule term describing S;
(e) W = S;
(f) Repeat b to e until the dataset contains only
    records of class i. The induced rule is then
    the conjunction of all the rule terms built at
    step d;
(g) Repeat a to f until all records of class i have
    been removed;
```

Cendrowska's original version of Prism requires the training set to be processed once for each class. It is restored to its original form before each new class is processed. A faster version of Prism has been developed by one of the present authors [6], called PrismTCS (**Prism** with **T**arget **C**lass, **S**mallest first) which maintains a similar level of predictive accuracy. After each rule induced PrismTCS computes the *target class* that covers the fewest instances and induces a new rule for that target class. Thus PrismTCS removes the outermost loop and lowers Prism's complexity. We removed the innermost loop, the multiple sorting of the dataset, by building attribute lists similar to those in the SPRINT algorithm [5, 7] of the structure <*record id, attribute value, class value*>.

Figure 1 shows experimental results with pre-sorting on the *diabetes* dataset [8]. We appended the data to itself either 'vertically' or 'horizontally' to obtain datasets with an increasingly large number of instances or attributes, respectively. We refer to data created by appending in the vertical direction as *portrait data* and to data created by appending in the horizontal direction as *landscape data*.

The speedup factors for all runs were positive, however decreasing for portrait data with increasing number of instances. However for data in landscape format the advantage of pre-sorting is increasingly linear.

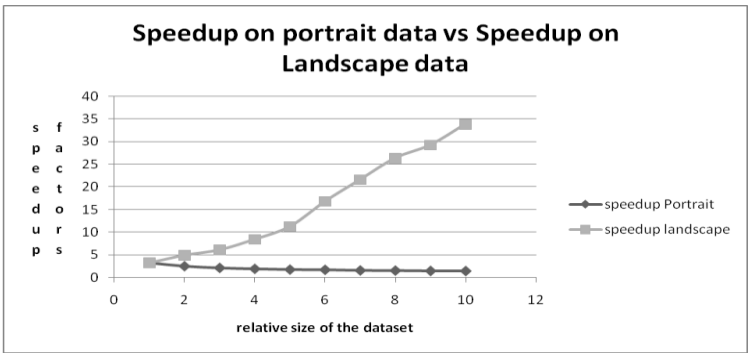

Figure 1 The speedup factors plotted versus the relative dataset size for datasets growing towards landscape and portrait format.

A version of PrismTCS incorporating the pre-sorting technique is currently being implemented. We expect that this will lead to higher speedups.

2. P-Prism: A Parallel Modular Classification Rule Induction Algorithm

There have been several attempts to scale up classification rule induction via parallelisation. In the area of TDIDT we have already mentioned the SPRINT [7] algorithm. Here we focus on parallelising modular classification rule induction using a "shared nothing" or "massively parallel processors" (MPP) system. Our reasoning is that MPP can be represented by a network of workstations and thus is a cheap way of running parallel algorithms. We implemented the parallel Prism (P-Prism) algorithm in a logical master worker fashion by running the basic Prism algorithm on a master machine and outsourcing the computationally expensive tasks, the induction of rule terms, to worker machines in the network. As a communication platform between the Prism algorithm and the worker machines we used a distributed blackboard system architecture based on the DARBS distributed blackboard system [9]. A blackboard system can be imagined as a physical blackboard which is observed by several experts with different knowledge domains, having a common problem to solve. Each expert uses its knowledge domain plus knowledge written on the blackboard to infer new knowledge about the problem and advertise it to the other experts by writing it on the blackboard. In the software model such a blackboard system can be represented by a client-server architecture. The basic architecture of P-Prism is shown in Figure 2.

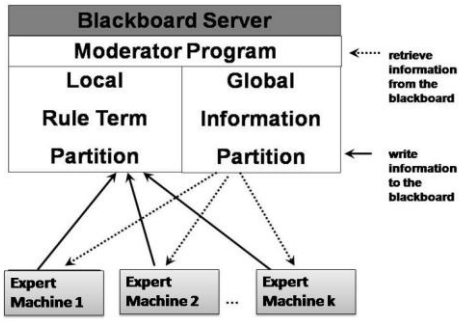

Figure 2. The architecture of the P-Prism algorithm using a distributed blackboard system in order to parallelise the induction of modular rule terms.

The attribute lists are distributed over k expert machines. The moderator program on the blackboard server implements the Prism algorithm, with the difference from the serial version that it delegates the rule term induction to the expert machines. The blackboard system is partitioned into two logical partitions, one to submit local rule term information and one to retrieve global information about the algorithm's status. Every expert is able to induce the rule term that is locally the best one for the attribute lists it holds. It then writes the induced rule term plus its covering probability and how many instances the rule term covers on the local rule term information partition and awaits the global information of how to continue.

The following steps listed below describe how P-Prism induces one rule:

```
Step 1 Moderator (P-Prism) writes on "Global Information Partition"
     the command to induce locally best rule terms.
Step 2 All Experts induce the locally best rule term and write the
     rule terms plus its covering probability and the number of
     list records covered on the "local Rule Term Partition"
Step 3 Moderator (P-Prism) compares all rule terms written on the
     "Local Rule Term Partition"; adds best term to the current
     rule; writes the name of the Expert that induced the best rule
     term on the Global Information Partition
Step 4 Expert retrieves name of winning expert.
     IF Expert is winning expert {
        derive by last induced rule term uncovered ids and write
        them on the "Global Information Partition" and delete
        uncovered list records
     }
     ELSE IF Expert is not winning expert {
         wait for by best rule term uncovered ids being available
         on the "Global Information Partition", download them and
         delete list records matching the retrieved ids.
     }
```

In order to induce the next rule term, P-Prism would loop back to step one. For P-Prism to know when to stop the rule it needs to know when the remaining list

records on the expert machines are either empty or consist only of instances of the current target class. This information is communicated between the winning expert and the moderator program using the Global Information Partition.

3. Experimental Results

The first prototype of the P-Prism classifier has been implemented and we have carried out an initial validation in order to identify constraints for future developments. We have carried out experiments with 3 different configurations of P-Prism and serial Prism with attribute lists. We used the *yeast* dataset [8] which comprises 1484 instances, but repeatedly appended the data to itself 'vertically' to achieve datasets with from 5000 to 35000 instances. Both versions of Prism, the serial and the parallel, produce identical rule sets (with 970 terms) on any of the yeast datasets.

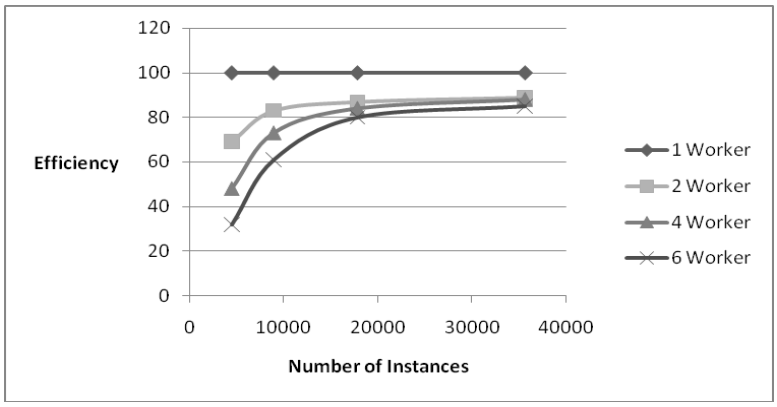

Figure 3 The efficiency calculated as the percentage of the actual speedup factors based on the ideal speedup factors for configurations of P-Prism with one (serial Prism), two, four and six expert machines.

The efficiency is the fraction of the actual speedup factor based on the ideal speedup factor of P-Prism. However achieving an ideal speedup factor is unrealistic as in any MPP environment we have to take overheads in bandwidth and workload balancing into account. The efficiencies for all P-Prism configurations increase with an increasing number of instances up to a workload of about 17000 instances. From then on the efficiency levels off, ranging from 85% to 89%. As already mentioned the discrepancy from 100% efficiency can be explained by communication overheads and workload balancing issues. However further speedup experiments are planned with more instances and more expert machines to examine the breakeven point of possible expert machines.

4. Ongoing and Future Work

Our experiments with P-Prism were based on the assumption that all expert machines have the same computational resources. This is not realistic and so we are currently working on an initial workload balancing strategy. All attribute lists will initially be advertised on a central server. Each expert will take an attribute list from the server, process it by scanning it for covering probabilities, and if there are more attribute lists left, it will take further ones until there are no attribute lists left. This will lead to an initial workload balancing as faster expert machines will retrieve more attribute lists from the scoreboard. A parallel version of PrismTCS based on P-Prism is also in development with which we hope to obtain better scale up results than those for P-Prism.

Our experiments with Prism show the value of the methods we have adopted. Prism has been used as an exemplar of an important class of rule generation algorithms, where each attribute can be processed independently of the others as rule terms are generated. Most rule covering algorithms are of this kind, including those for generalized rule induction (where the right-hand side of each rule can potentially be a conjunction of attribute/value pairs for any combination of categorical attributes) as well as classification.

References

1. Catlett J., *Megainduction: Machine learning on very large databases*. 1991, University of Technology, Sydney.
2. Quinlan J. R., *Induction of decision trees. Machine Learning*. Vol. 1. 1986. 81-106.
3. I.Witten and E.Frank. Data Mining: Practical Machine Learning Tools and Techniques. Elsevier, 2005.
4. Cendrowska J., *PRISM: an Algorithm for Inducing Modular Rules.* International Journal of Man-Machine Studies, 1987. **27**: p. 349-370.
5. Stahl F. and Bramer M., *Towards a Computationally Efficient Approach to Modular Classification Rule Induction.* Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, 2007.
6. Bramer M., *An Information-Theoretic Approach to the Pre-pruning of Classification Rules*. Proceedings of the IFIP Seventeenth World Computer Congress - TC12 Stream on Intelligent Information Processing. 2002: Kluwer, B.V. 201-212.
7. Shafer J. C., Agrawal R., and Mehta M., *SPRINT: A Scalable Parallel Classifier for Data Mining.* Twenty-second International Conference on Very Large Data Bases, 1996.
8. Blake C. L. and Merz C. J, *UCI repository of machine learning databases*. 1998, University of California, Irvine, Department of Information and Computer Sciences.
9. Nolle L., Wong K. C. P., and Hopgood A., *DARBS: A Distributed Blackboard System.* Twenty-first SGES International Conference on Knowledge Based Systems, 2001.