J-PMCRI: A Methodology for Inducing Pre-Pruned Modular Classification Rules

Frederic Stahl, Max Bramer, and Mo Adda

University of Portsmouth, School of Computing,
Buckingham Building, Lion Terrace, Portsmouth PO1 3HE, United Kindom
{Frederic.Stahl,Max.Bramer,Mo.Adda}@port.ac.uk
http://www.port.ac.uk

Abstract. Inducing rules from very large datasets is one of the most challenging areas in data mining. Several approaches exist to scaling up classification rule induction to large datasets, namely data reduction and the parallelisation of classification rule induction algorithms. In the area of parallelisation of classification rule induction algorithms most of the work has been concentrated on the Top Down Induction of Decision Trees (TDIDT), also known as the 'divide and conquer' approach. However powerful alternative algorithms exist that induce modular rules. Most of these alternative algorithms follow the 'separate and conquer' approach of inducing rules, but very little work has been done to make the 'separate and conquer' approach scale better on large training data. This paper examines the potential of the recently developed blackboard based J-PMCRI methodology for parallelising modular classification rule induction algorithms that follow the 'separate and conquer' approach. A concrete implementation of the methodology is evaluated empirically on very large datasets.

Key words: Parallel Rule Induction, Blackboard Systems, Modular Rule Induction.

1 Introduction

Parallel Data Mining is motivated by the need to scale up to massive datasets. The runtime complexity of data mining is typically linear or worse with respect to both the number of instances and the number of attributes in a dataset, thus massive datasets can be computationally expensive to analyse. There are many examples of applications that need to deal with the problem of mining massive datasets. For example, in bioinformatics and chemistry there are large datasets which are generated in different kinds of Molecular Dynamics (MD) simulations and there is a considerable desire to find ways to manage, store and find complex relationships in this generated data [2, 14]. Other examples of massive datasets are those in the area of astronomy for example generated by the NASA system of earth orbiting satellites and other space-borne probes launched in 1991 and still ongoing [17]. A further example in astronomy is the Sloan survey [16].

In the area of classification rule induction most of the approaches to scaling up the algorithms by parallelisation have concentrated on the Top Down Induction of Decision Trees (TDIDT), also known as the 'divide and conquer' approach. Notable parallel 'divide and conquer' algorithms and systems are SLIQ [9], SPRINT [11] and their successor, the ScalParC [8] algorithm. For building split points of attributes, SLIQ uses a sorted attribute list for each attribute of the form <*record id, attribute value*>, and a class list of the form <*class value, tree node*>. Each processor is assigned to a sublist of each attribute list plus the class list. Different to SLIQ, SPRINT and ScalParC build a sorted attribute list for each attribute of the form <*record id, attribute value, class value*> and no class list. Again each processor is assigned to a sublist of each attribute list and exchanges the statistics needed in order to determine the best split point. Each processor builds the tree simultaneously and updates the attribute lists accordingly.

On the other hand there have been virtually no attempts to parallelise algorithms that induce modular rules rather than decision trees. Modular rules are rules that do not necessarily fit into a decision tree. Most modular classification rule induction algorithms follow the 'separate and conquer' approach. The Parallel Modular Classification Rule Induction (PMCRI) methodology has recently been developed in order to provide a methodology that allows the parallelisation of a whole subset of modular classification rule induction algorithms. In this work we will outline the PMCRI methodology and highlight an extension of it, that integrates pre-pruning, the J-PMCRI methodology. J-PMCRI is designed to run in a local area Network (LAN) in order to provide an inexpensive data mining solution for large datasets for modest sized organisations that cannot afford supercomputers. This work provides a concrete implementation of the J-PMCRI methodology that incorporates pre-pruning and is presented and evaluated empirically.

2 J-PMCRI

The PMCRI/J-PMCRI methodology is designed to parallelise any member of the Prism family of algorithms. Section 2.1 will give a brief introduction to the Prism family and Section 2.2 will describe J-PMCRI.

2.1 Prism Family of Algorithms

Cendrowska's criticism [7] of a tree representation of classification rules is that they do not directly allow the induction of modular rules such as:

$$IF\ a = 1\ and\ b = 1\ then\ class = A$$
$$IF\ c = 1\ and\ d = 1\ then\ class = B$$

Such rules do not necessarily have attributes in common in their rule terms unlike for the representation in tree format. Inducing such rules will lead to

the replicated subtree problem, first described by [7]. Prism has been examined empirically and has shown to be comparable to decision trees and in some cases even shows a better predictive performance [3]. Subsequent research developed further variations of Prism algorithms notably the PrismTCS algorithm and PrismTC [4, 5]. The theoretical worst case complexity of Prism algorithms is $O(N^2 \cdot M)$ where N is the number of data instances and M is the number of attributes. However an empirical examination revealed an average complexity of $O(N \cdot M)$ [15].

Our implementation of the basic Prism approach for continuous data only is summarised in the following pseudo code: where A_x is a possible attribute value and D is the training dataset.

```
For each class i do {
  Step 1: Calculate for each Ax probabilities p(class = i| A < Ax) and p(class = i| A >= Ax)
  Step 2: Select the condition with the maximum probability as rule term
          and create a subset D' of D that comprises all instances
          that match the selected condition.
  Step 3: Repeat 1 to 2 for D' until D'  only contains instances
          of classification i. The induced rule is then a
          conjunction of all the selected conditions and i.
  Step 4: Create a new D' that comprises all instances of D except
          those that are covered by the rules induced for class i so far.
  Step 5: Repeat steps 1 to 4 until D' does not contain any
          instances of classification i.
}
```

J-pruning, a pre-pruning method, developed by Bramer [4], demonstrated good performance with respect to the predictive accuracy of the classifier[4, 6]. J-pruning is based on the J-measure from Smyth and Goodman [13] and can be applied to TDIDT and any member of the Prism family. Also it has been observed in [15] that J-pruning lowers the number of rules and rule terms induced and thus the number of iterations of the algorithm, which in turn lowers the runtime.

According to Smyth and Goodman [13] the average information content of a rule of the form $IF\ Y\ =\ y\ THEN\ X\ =\ x$ can be quantified by the following equation:

$$J(X;Y=y) = p(y) \cdot j(X;Y=y) \quad (1)$$

The J-measure is a product of two terms. The first term *p(y)* is the probability that the antecedent of the rule will occur. It is a measure of the hypothesis simplicity. The second term *j(X;Y=y)* is the j-measure or cross entropy. It is a measure of the goodness-of-fit of a rule and is defined by:

$$j(X;Y=y) = p(x \mid y) \cdot log(\tfrac{p(x|y)}{p(x)}) + (1 - p(x \mid y)) \cdot log(\tfrac{(1-p(x|y))}{(1-p(x))}) \quad (2)$$

If a rule has a high J-value then it tends to have a high predictive accuracy as well. The J-value is used to identify when a further specialisation of the rule is likely to result in a lower predictive accuracy due to overfitting. The basic idea is to induce a rule term and if the rule term would increase the J-value of the current rule then the rule term is appended. If not then the rule term is discarded and the rule is finished.

2.2 The J-PMCRI Methodology

Both the PMCRI and J-PMCRI methodology distribute the workload of inducing rule terms over a network of workstations in a LAN by distributing the training data. The basic rule induction procedure of the J-PMCRI can be divided into three steps:

1. the training data is distributed over n workstations
2. learning algorithms on each workstation cooperate to induce a rule and communicate in order to get a global view of the state of the classifier
3. combine the local parts of the rule to a final classifier.

With regards to *step 1*, a workload balance is achieved by building attribute lists out of each attribute in the training data similar to those in the SPRINT [11] algorithm. Attribute lists are of the structure <*record id, attribute value, class value*>. These attribute lists are then distributed evenly over n workstations. Unlike SPRINT, which achieves a workload balance by splitting each attribute list into n sublists and each workstation gets assigned a part of each attribute list. We distribute entire attribute lists evenly over all workstations. Distributing parts of the attribute lists may achieve a better workload balance at the very beginning, however it is likely that it will result in a considerable workload imbalance later on in the algorithm's execution as part attribute lists may not evenly decrease in size [12]. Distributing entire attribute lists may only impose a slight workload imbalance at the beginning of the algorithm in J-PMCRI, however the relative workload on each workstation will approximately stay the same. With regards to *step 2* every workstation holds a subset of the attribute lists and thus a subset of the feature space of the training data in memory. Each workstation can induce the conditional probabilities for candidate rule terms for their attribute lists independently and thus can derive a candidate rule term that is locally the best for the attribute lists in the workstation's memory. However in order to find out which is the globally best rule term the workstation needs to exchange information about the locally induced rule terms with the other workstations. For this purpose we use a distributed blackboard architecture like the one in [10]. A blackboard architecture can be seen as a physical blackboard, that is observed and used by several experts with different knowledge domains that have a common problem to solve. Each expert will use its own knowledge plus information written on the blackboard by other experts in order to derive new information and write it on the blackboard. As a software model this principle can be represented by a client server architecture. In the blackboard literature these experts are also called Knowledge Sources (KS). The basic architecture of J-PMCRI is shown in figure 1. The attribute lists are distributed over k KS machines in the network. The blackboard system is partitioned into two logical panels or partitions that have a different meaning to the KS machines, one for information about local rule terms, on the 'local rule term information' (LI) partition and one for global information (GI).

Every KS is hosted on a separate machine in the network and is able to induce the rule term that is locally the best one for the attribute lists it holds

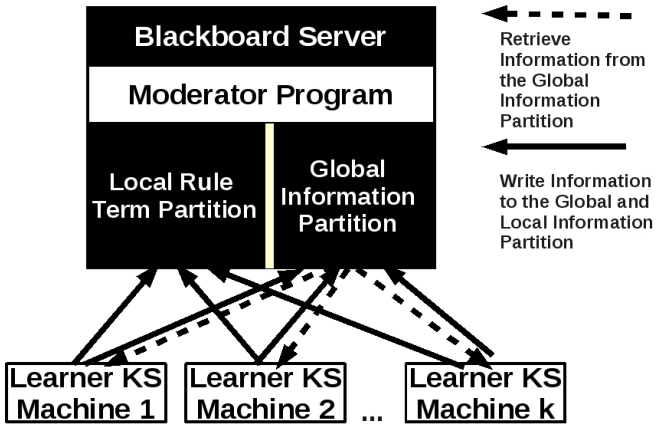

Fig. 1. PMCRI's communication pattern using a distributed blackboard architecture.

in memory. It then writes information about the induced rule term on the LI partition and next it awaits the global information it needs in order to induce the next rule term being advertised on the GI partition. The information submitted about the rule term is the probability with which the induced rule term covers the target class on the local attribute list collection. The moderator program which is embedded in the blackboard server collects the probabilities from the LI partition, compares them and advertises the name of the KS that induced the globally best rule on the GI partition. The KSs that have only induced a locally best rule term will delete the term from their memory. The KS with the globally best rule term will keep the rule term in memory and then communicate the ids of the instances that are uncovered by this rule term to the other waiting KS, using the GI partition. Now the next rule term can be induced in the same way.

The information needed to calculate the J-value as outlined in formulae 1 and 2 in Section 2.1, is the count of how many data instances (list instances) the rule term covers, the count of how many instances covered by the rule term are assigned with the target class, the total number of instances and the total number of instances covering the target class. This information is solely contained in the attribute lists the locally best rule term was induced from. The basic idea is to not only write the probability with which the rule term was induced but also the J-value of the rule, if the rule term would be appended, on the LI partition. The moderator first compares the probabilities and thus identifies the KS that has induced the globally best rule term. Next the moderator compares the new J-value with the previous one. If the new J-value is larger than the winning KS, it is advertised on the GI partition otherwise the moderator writes on the 'global information partition' that the rule needs to be pruned. The KS will act accordingly. They will discard the last locally induced rule term and start inducing the first candidate rule term for the next rule. The moderator pseudo code is shown below:

```
bestJ=0; bestProb=0; ExpertInfo;
for each submitted rule term do{
  IF(t.p>bestProb){
     if(t.j>bestJ){
        ExpertInfo = Best term induced on  t.ExpertName;}
     else{
        ExpertInfo = prune rule;}
   }
}
```

The following pseudo code describes how a KS induces rules based on the Prism algorithm outlined in Section 2.1. However it can easily be adapted to any other member of the Prism family.

```
Rule_Set{};
For each class i do{
  Step 1: Initialise empty rule r;
  Step 2: Calculate for each attribute list Ax p(class = i| A < Ax)
          and p(class = i| A >= Ax) and the corresponding J-value and coverage;
  Step 3: Select the condition with the maximum probability as locally best
          rule term t;
  Step 4: Write KS name plus p, J-value and coverage on the LI partition
          and observe the GI partition for information I:
          IF(I equals own name){
            add t to r;
            generate from t uncovered Ids and communicate them to the remaining KS;
            delete attribute list instances that match the uncovered Ids;
          }ELSE IF(I = "prune rule"){
            add r to Rule_Set;
          }ELSE (I equals name of different KS){
            retrieve Ids that are uncovered from the globally best rule term;
            delete attribute list instances that match the uncovered Ids;
          }
  Step 4: Restore all attribute lists to their initial size;
          delete all instances from all attribute lists that are covered by Rule_Set{};
} While still instances left in the training data
```

At the end of J-PMCRI's execution the KS will only have the rule terms in their memory that they induced locally and that were globally the best ones. They simply communicate the rule terms plus information about which rule and which class they were induced for to the blackboard server. There the globally best rule terms are assembled to the final rules.

The Prism algorithm parallelised J-PMCRI would induce exactly the same rules as the serial version would using J-pruning, only they are parallel and faster to generate and thus can computationally cope with much larger data volumes.

3 Evaluation of the J-PMCRI

Previous evaluations of the methodology have focussed mainly on the size up behaviour, which measures the runtimes of the system with respect to the number of training instances or data records. For both the number of training instances and data records in previous work a slightly better than linear size up behaviour has been observed, meaning that if the amount of data doubles then the runtimes nearly double as well, but not quite. These experiments have been performed with several smaller datasets from the UCI repository that have been appended to themselves in order to increase the number of instances and attributes and

thus the workload. The reason for this was to keep the number of rules and rule terms induced constant, so that they will not influence the runtime severely, as a different number of rules and rule terms results in a different number of iterations of the algorithm and thus in a different runtime.

In this work we evaluate the J-PMCRI methodology with respect to the number of processors (workstations) used assuming that there is only one KS hosted on each workstation. Also we will evaluate this on two real large datasets from the infobiotics repository [1], which comprises very large datasets for benchmarking purposes. The first dataset is called (here) the infobio2 dataset. It comprises 60 attributes, 4 classifications and more than 2.3 million continuous training instances. The second dataset used is called (here) infobio3 and comprises 220 attributes, 4 classifications and 2.5 million continuous data instances. A standard metric to evaluate a parallel algorithm or a parallel architecture is the speedup factor. With the speedup factors one can calculate by how much the parallel version of an algorithm is faster using p processors compared with 1 processor.

$$S_p = \frac{R_1}{R_p} \qquad (3)$$

Formula 3 represents the speedup factors S_p. R_1 is the runtime of the algorithm on a single machine and R_p is the runtime on p machines. In the ideal case, the speedup factors are the same as the number of processors used. In reality, the speedup factor will be below the number of processors for various reasons, such as communication and synchronisation overheads imposed by each processor.

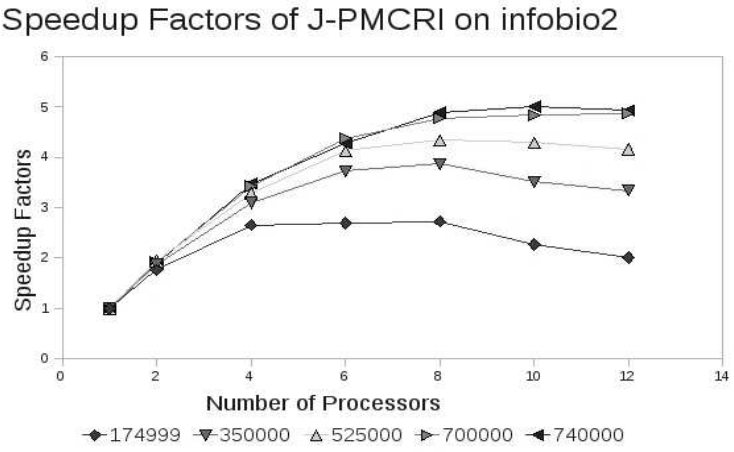

Fig. 2. Speedup factors obtained for J-PMCRI with the PrismTCS algorithm on the infobio2 datasets.

Figure 2 shows speedup factors plotted versus the number of processors used on fixed numbers of training instances of the infobio2 dataset. The speedup

factors increase with an increasing number of processors then decrease again. This can be explained by the fact that using more processors will impose a larger communication overhead as well as managing overhead. However, what can also be observed is that the best speedup is reached for a larger number of processors if the number of training instances is large as well. Thus, loosely speaking, the larger the number of training instances the more processors are beneficial. Please note that the experiments illustrated in figures 2 and 3 do not include an experiment with all data instances. The reason for this is that it was not possible to perform this experiment on a 1 processor configuration due to memory constraints.

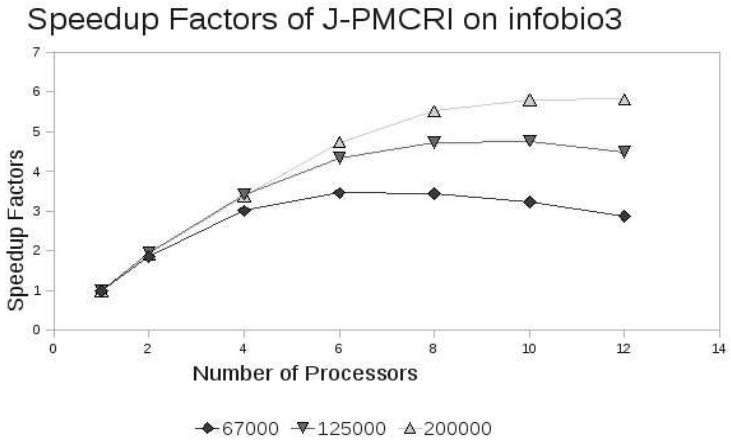

Fig. 3. Speedup factors obtained for J-PMCRI with the PrismTCS algorithm on the infobio3 datasets.

Figure 3 shows similar experiments as in figure 2 only this time on the infobio3 dataset in order to show that the results are reproducible.

Figure 4 shows the speedup factors obtained on a much larger portion of the training data compared with the experiments in figures 2 and 3. The experiments on infobio2 comprised a total of 2,338,121 (and 60 attributes) instances and on inforbio 3 a total of 700,336 (and 220 attributes). The reason for these experiments is to show that J-PMCRI would perform well with many processors if the data were large enough. For the calculation of the speedup factors the system needs to be based on a configuration with fewer processors than the observed configurations. In the ideal case of one processor, however, the training data is too large to fit in the memory of a single workstation. We based the speedup factors on a 4 processor configuration. The results are shown in figure 4. What can be seen in figure 4 is that the speedup factors are still increasing with a total of 12 learner KS machines. This is different compared with the experiments outlined in figures 2 and 3, where 12 learner KS machines were not beneficial

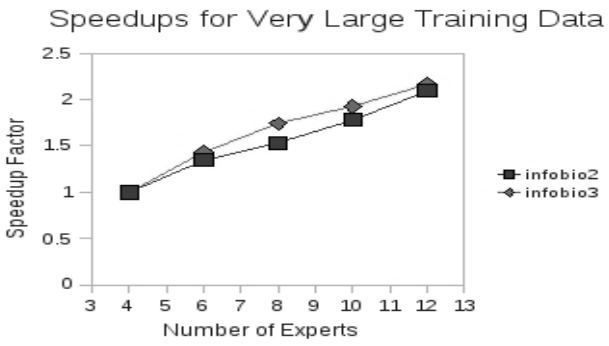

Fig. 4. Speedups obtained using J-PMCRI on very large training data that cannot be handled by a single machine.

anymore. The fact that the speedup factors are still growing for 12 processors highlights even more the observed behaviour that using more processors is more beneficial the larger the amount of training data.

4 Conclusions

This paper has addressed the need for better scaling classification rule induction algorithms. The argument established in the introduction was that there are many approaches to speeding up the 'divide and conquer' approach which induces decision trees. But there are virtually no attempts to scale up the 'separate and conquer' approach for inducing modular rules. This work's aim is to establish a methodology for parallelising and thus scaling up algorithms of the Prism family that induce modular rules. The methodology is outlined together with a pre-pruning facility and pseudo code is given in order to illustrate the basic concept. The system is based on a blackboard approach, can be run in a LAN and has been evaluated with respect to the number of processors (workstations) used. Using J-PMCRI has the very desirable property that the optimum number of processors increases as the amount of training data becomes larger.

Ongoing and future work comprises the implementation of further algorithms of the Prism family using the J-PMCRI methodology. Also the speedup factors illustrated in Section 3 may be improved by using more than just one blackboard. For instance the connection in the LAN (assuming a switched network) that connects the Blackboard server with the LAN will be used by all KS. However if further blackboard servers are introduced each blackboard only needs to serve a subset of the KS machines. These Blackboard servers will need to be synchronised as well, but this could easily be established using a separate smaller network for synchronising the blackboards.

References

1. J. Bacardit and N. Krasnogor. The infobiotics PSP benchmarks repository. Technical report, 2008.
2. D. Berrar, F. Stahl, C. S. G. Silva, J. R. Rodrigues, and R. M. M. Brito. Towards data warehousing and mining of protein unfolding simulation data. *Journal of Clinical Monitoring and Computing*, 19:307–317, 2005.
3. M. Bramer. Automatic induction of classification rules from examples using N-Prism. In *Research and Development in Intelligent Systems XVI*, pages 99–121, Cambridge, 2000. Springer-Verlag.
4. M. Bramer. An information-theoretic approach to the pre-pruning of classification rules. In B Neumann M Musen and R Studer, editors, *Intelligent Information Processing*, pages 201–212. Kluwer, 2002.
5. M. Bramer. Inducer: a public domain workbench for data mining. *International Journal of Systems Science*, 36(14):909–919, 2005.
6. M. Bramer. *Principles of Data Mining*. Springer, 2007.
7. J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.
8. M.V. Joshi, G. Karypis, and V. Kumar. Scalparc: a new scalable and efficient parallel classification algorithm for mining large datasets. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 573–579, 1998.
9. M. Metha, R. Agrawal, and J. Rissanen. SLIQ: a fast scalable classifier for data mining. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, volume 1057, pages 18–32. Springer, 1996.
10. L. Nolle, K. C. P. Wong, and A. Hopgood. DARBS: a distributed blackboard system. In *Proceedings of the Twentyfirst SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, Cambridge, 2001. Springer.
11. J. Shafer, R. Agrawal, and M. Metha. SPRINT: a scalable parallel classifier for data mining. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, pages 544–555. Morgan Kaufmann, 1996.
12. A. Sirvastava, E. Han, V. Kumar, and V. Singh. Parallel formulations of Decision-Tree classification algorithms. *Data Mining and Knowledge Discovery*, pages 237–261, 1998.
13. P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases. *Transactions on Knowledge and Data Engineering*, 4(4):301–316, 1992.
14. F. Stahl, D. Berrar, C. S. G. Silva, J. R. Rodrigues, and R. M. M.s Brito. Grid warehousing of molecular dynamics protein unfolding data. In *Proceedings of the Fifth IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, pages 496–503, Cardiff, 2005. IEEE/ACM.
15. F. Stahl, M. Bramer, and M. Adda. Parallel rule induction with information theoretic pre-pruning. In *Research and Development in Intelligent Systems XXV*, pages 151–164, Cambridge, 2010. Springer.
16. A Szalay. *The Evolving Universe*. ASSL 231, 1998.
17. J. Way and E. A. Smith. The evolution of synthetic aperture radar systems and their progression to the EOS SAR. *IEEE Transactions on Geoscience and Remote Sensing*, 29(6):962–985, 1991.