Parallel Rule Induction with Information Theoretic Pre-Pruning

Frederic Stahl, Max Bramer and Mo Adda[1]

Abstract In a world where data is captured on a large scale the major challenge for data mining algorithms is to be able to scale up to large datasets. There are two main approaches to inducing classification rules, one is the *divide and conquer* approach, also known as the top down induction of decision trees; the other approach is called the *separate and conquer approach*. A considerable amount of work has been done on scaling up the divide and conquer approach. However, very little work has been conducted on scaling up the *separate and conquer approach.* In this work we describe a parallel framework that allows the parallelisation of a certain family of separate and conquer algorithms, the Prism family. Parallelisation helps the Prism family of algorithms to harvest additional computer resources in a network of computers in order to make the induction of classification rules scale better on large datasets. Our framework also incorporates a pre-pruning facility for parallel Prism algorithms.

1 Introduction

Induction of classification rules from data samples in order to predict previously unseen data can be traced back to the 1960s [1]. The two most popular approaches to classification rule induction are the *divide and conquer* approach and the *separate and conquer* approach. The *divide and conquer* approach induces classification rules by recursively breaking down the classification problem into sub-problems. The resulting rules are in the form of decision trees and thus *divide and conquer* is also known as the Top Down Induction of Decision Trees (TDIDT) [2]. TDIDT resulted in a wide range of classifiers such as the C4.5 and the C5.0 systems. The *separate and conquer* approach directly searches for a rule that explains a part of the training data, separates the part of the data that is covered by the rule and recursively searches for new rules on the remaining examples until there are no training instances left. *Separate and conquer* can be traced back to the 1960s to the AQ learning system [3]. Rule induction algorithms based on the *separate and conquer* approach often produce more general rules compared with decision trees on noisy training data. Notably the Prism algorithm [4] often produces qualitatively better rules than TDIDT especially on noisy data.

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

M. Bramer et al. (eds.), *Research and Development in Intelligent Systems XXVI*,
DOI 10.1007/978-1-84882-983-1_11,

The fast development in processing power, storage and sensor technology, notably CCTV cameras leads to the generation and storage of larger datasets. However, researchers still wish to apply data mining algorithms such as classification rule induction algorithms to large datasets. There are two general approaches to scaling up classification rule induction algorithms; sampling and the development of parallel classification rule induction algorithms. Sampling the data before the classification rule induction algorithm is applied has been criticised by Catlett [5] who showed that the accuracy of an induced classifier increases with an increasing size of the training sample. However, Catlett conducted his research 18 years ago and datasets that were considered to be large in his work are commonplace today. In 1999 Frey and Fisher justified sampling by showing that the rate of increase of the accuracy of the classifier slows down with the rate of increase of the training data. However, applications that demand high classification accuracy and applications that are concerned with the discovery of new knowledge or where the data size is simply so large that even sampled versions are massive in size still desire scalable classification rule induction technology. The development of parallel classification rule induction algorithms has been concentrated on the TDIDT approach, notably by the SLIQ algorithm [6] and its successor SPRINT [7]. SPRINT claims to achieve a linear scale up with the increase of the training data; however Srivastava pointed out that the breath first search approach that SPRINT uses might result in workload balancing problems during its execution [8]. In general concerning the predictive accuracy, only parallel versions of decision tree induction algorithms showed an acceptable performance. The only attempt to parallelise algorithms of the *separate and conquer* approach to date is the Parallel Modular Classification Rule Induction (PMCRI) project [9]. The PMCRI framework has been constructed to parallelise algorithms of the Prism family. In this paper we present an implementation of PMCRI with the extension of a pre-pruning facility.

2 Modular Classification Rule Induction With Prism

The development of Prism is a result of the main criticism of TDIDT, which is the intermediate representation of classification rules in the form of a tree [4]. A tree representation of classification rules does 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 common attributes in their rule terms unlike for the representation in tree format. Thus the induction of decision trees will produce unnecessarily large and confusing rule sets. Cendrowska's Prism algorithm induces modular rules. In subsequent studies Prism has also been shown to be less vulnerable to clashes. However, the Prism algorithm does not scale well

on large datasets. A version of Prism that attempts to scale up Prism to larger datasets is the PrismTCS (**Prism** with **T**arget **C**lass, **S**mallest first) algorithm [10] which has been developed by one of the authors. PrismTCS has a comparable level of predictive accuracy to Prism. The only difference between the two algorithms is that whereas in Prism, the above described *separate-and-conquer* approach is applied for each class value in turn, in PrismTCS it is only applied once.

Our implementation of PrismTCS for continuous data only is summarised in the following pseudo code:

```
(a) working dataset W = restore Dataset;
    delete all records that match the rules
    that have been derived so far;
    target class i = class that covers the
    fewest instances in W;
(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) restore Dataset = restore Dataset – W;
    Repeat a to f until W only contains
    instances of class i or is empty;
```

However there is a whole family of Prism algorithms besides Prism and PrismTCS. There is also PrismTC, which differs from PrismTCS in that it selects as target class the class that covers the most instances, rather than the least [11].

2.1 Pre-Pruning of Prism Classification Rules

Classifiers are usually pruned in order to reduce overfitting of classification rules. There are two general types of pruning, *post-pruning* and *pre-pruning*. Post-pruning methods are applied to the already trained classifier whereas pre-pruning is applied as the rules are being generated. Most parallel versions of TDIDT classifiers follow the post pruning approach with the reasoning that it takes only a small fraction of the overall induction time of the classifier [7, 8]. However, pre-pruning generally leads to slimmer classifiers and thus reduces the number of iterations of the algorithm and thus the classification rule induction time. A pre-pruning method introduced by Bramer [10] based on the J-measure of Smyth and Goodman [12] can be applied to both the TDIDT and the Prism family of

algorithms and it shows on both families a good performance [10] with respect to predictive accuracy and the number of rule terms. Thus we will incorporate J-pruning in our PMCRI framework.

According to [12] a rule of the form *IF Y=y, THEN X=x* has the average information content of:

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

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:

$$j(X;Y=y) = p(x \mid y).\log_2(\frac{p(x \mid y)}{p(x)}) + (1-p(x \mid y)).\log_2(\frac{(1-p(x \mid y))}{(1-p(x))})$$

For further reading we refer to [12]. For J-pruning it is assumed that a rule having a high J-value will tend to have a high predictive accuracy. So the J-measure can be used to identify a point where a further specialisation of a rule is likely to become counter productive because of overfitting. In Prism J-pruning is performed by measuring the J-value of each rule after appending a rule term. If the J-value increases then the term is accepted otherwise the term is rejected and a clash handling procedure is invoked. If a rule is pruned Prism calculates the majority class in the subset. If it is not the target class then the entire rule is discarded and instances of the training subset that belong to the target class are deleted.

2.2 Scalability of Prism

We derived the theoretical complexity of Prism based on the number of calculations of the probability for a possible split value of an attribute contained in step *b* in the PrismTCS pseudo code above. We will call this number the number of cutpoints. In the best case scenario there would be only one attribute that determines the class of any data instance or all data instances would simply belong to the same class. It is difficult to describe an average case for the algorithm as the outcome is strongly dependent on the underlying pattern in the data. However, it is possible to describe the worse case. Let *N* be the number of data instances and *M* the number of attributes. A categorical attribute will at most occur once in a rule whereas a continuous one may occur twice, as with two rule terms it would be possible to describe any interval of values of a continuous attribute. Thus there will be a maximum number of *2M* rule terms per rule. The maximum number of rules is *N-1*, meaning that each training instance except one is described by a separate rule. The *-1* is because if there is only one instance left in the training data we do not need to generate a further rule for it. The complexity of inducing the *kth* rule is *2M(N-k)*. For example if we induce the very first rule *(k=1)* we

would have *N* instances available thus we would have *(N-1)* cutpoint calculations per rule term. As there are a maximum of *2M* rule terms there would be altogether *2M(N-1)* cutpoint calculations. Summing this up for all possible rules leads to:

$$\sum_{k=1}^{N-1}(2M)\cdot(N-k) = 2M\cdot\frac{N\cdot(N-1)}{2} = O(N^2M)$$

A complexity of $O(N^2M)$ is very pessimistic and seems to be very unlikely to happen. In practice larger datasets may well contain fewer rule terms than smaller ones. Also J-pruning will reduce the number or rule terms induced as shown in [10].

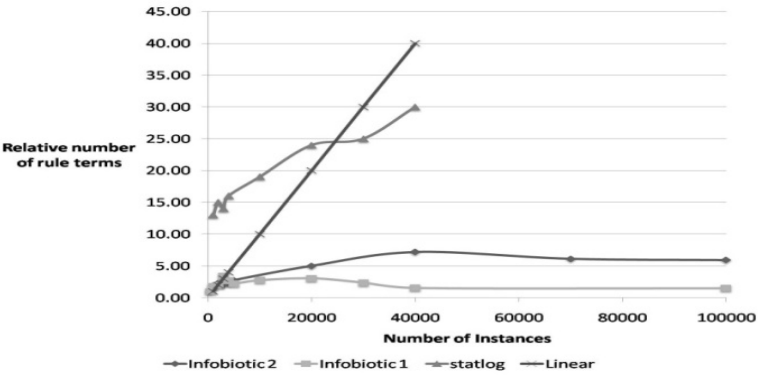

Fig. 1. Relative number of rule terms induced versus the size of the training data.

We ran PrismTCS with J-Pruning on several datasets. For each dataset we built samples of different sizes and measured the number of rule terms induced. We plotted the relative number of rule terms versus sample size in figure 1. The Statlog (Shuttle) dataset is from the UCI repository [13] and the infobiotics 1 and 2 datasets were retrieved from the infobiotics data repository which comprises large real world datasets for benchmark tests for machine learning tasks [14]. The Statlog dataset samples ranged from 1000 to 40000 instances and comprised 9 attributes and 5 classes; infobiotics 1 ranged from 1000 to 100000 instances comprising 20 attributes and 5 classes, infobiotics 2 also ranged from 1000 to 100000 instances but comprised 100 attributes and 5 classes. Figure 1 also contains the relative number and theoretical linear relative number of rule terms plotted versus the number of training instances and shows that there is no linear behaviour of the increase of the number of rule terms recognizable. The scaling results obtained in [9] strongly suggest a linear scaling behaviour of the Prism family with respect to the number of training instances while the number of rules and rule terms remains constant.

3 J-PMCRI: Parallel Modular Classification Rule Induction With J-Pruning

The basic idea is to distribute the workload of Prism over a network of computers by distributing the training data. Each computer in the network does its part in inducing the classifier. To realise such a loosely coupled system no special hardware is required. In PMCRI the classifier induced is exactly the same as would be induced from the serial version of the Prism algorithm.

3.1 Architecture of PMCRI

The architecture of PMCRI is based on the Cooperating Data Mining model (CDM) [15]. The CDM model can be divided into three layers; the first comprises a sample selection procedure; in the second layer learning algorithms work on the local training data and communicate in order to get a global view of the state of the classifier; the third layer is a combining procedure that assembles the final classifier using rule terms induced locally on all machines. In the first layer a workload balance is achieved by building attribute lists out of each attribute in the training data similar to those in the SPRINT [7] algorithm. Attribute Lists are of the structure *<record id, attribute value, class value>*. These attribute lists are then distributed evenly over *n* processors. Unlike SPRINT, which achieves a workload balance by splitting each attribute list into *n* chunks and distributes them evenly over *n* processors we distribute entire attribute lists evenly over *n* processors. 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 as part attribute lists may not evenly decrease in size [8]. Distributing entire attribute lists may only impose a slight workload imbalance at the beginning of the algorithm in PMCRI. However the relative workload on each processor will approximately stay the same. Now having distributed the entire training data in the form of attribute lists, each processor will be able to induce a rule term, which is locally the best rule term for the attribute lists it holds in memory. Once each rule term is induced the participating machines need to exchange information in order to find out which one induced the globally best rule term. For this purpose we use a distributed blackboard architecture as in [16]. 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 in turn write it on the blackboard. As a software model this principle can be represented by a client server architecture. The basic architecture of PMCRI is shown in figure 2 [17]. The attribute lists are distributed over *k* machines in the network. The blackboard system is partitioned

into two logical partitions, one for information about local rule terms on experts and one for global information.

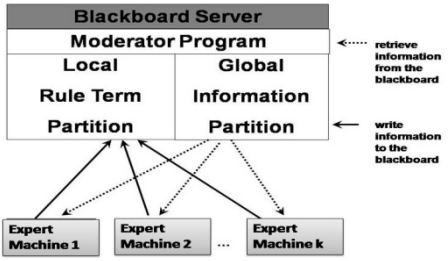

Fig. 2. The architecture of the PMCRI framework using a distributed blackboard architecture in order to parallelise any member of the Prism Family.

Every expert 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. It then writes information about the induced rule term on the local rule term information partition and awaits the global information it needs in order to induce the next rule term. The information submitted is basically the covering probability with which the induced rule term covers the target class on the local attribute list collection and the number of instances this rule term covers. If the local rule term information is submitted from all *k* expert machines the moderator program on the blackboard server will collect this information and use it in order to determine the globally best rule term. The moderator advertises the winning expert to all experts by writing the winning expert's name on the global information partition. The winning expert then will communicate the ids of the instances that are uncovered by this rule term to the other waiting experts using the blackboard system. Now the next rule term can be induced in the same way.

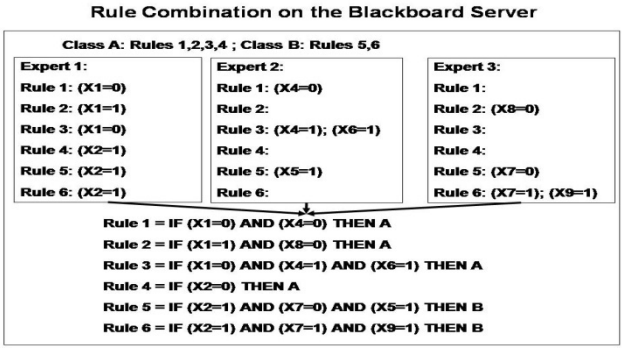

Fig. 3. Combining Procedure in PMCRI

At the end of the PMCRI execution each expert machine will hold a set of terms for each rule. The implementation of the combining procedure in layer three in the CDM model is realised by communicating all the rule terms locally stored at the expert machines to the blackboard. Each rule term is associated with information about the rule and the class for which the terms were induced. The moderator program then simply appends each rule term to its corresponding rule as illustrated in figure 3.

3.2 Parallel J-Pruning in PMCRI

The following steps listed below describe how PMCRI induces one rule [18] based on the Prism algorithm:

```
Step 1 Moderator (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 (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, PMCRI would loop back to step one. For PMCRI 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.

J-pruning in Prism can be integrated in step 2 and 3 in the algorithm above and thus no further synchronisation is needed. That is because the attribute list from which the local rule term on an expert was induced contains enough information to calculate the J-value for the term concerned. The information needed to calculate the J-value 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 that are assigned with the target class, the total number of instances and the total number of instances covering the target class. Each expert calculates the J-value in step 2 and writes it on the blackboard. Then in step 3 of the moderator program the best rule term location can be determined by the following procedure if *p* is the covering probability, *c* the number of instances covered assigned with the target class, *j* being the rule terms J-value:

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

The Moderator will write the content of "ExpertInfo" to the Global Information Partition on the blackboard. If the message contains the name of the winning expert the algorithm will continue with step 4, if it contains the info "prune rule" then each expert will invoke the clash resolution procedure outlined in section 2.1 and start the next rule.

4 Evaluation of the PMCRI Framework

In order to evaluate PMCRI we used the diabetes and yeast datasets from the UCI repository [13]. To create a larger and thus more challenging workload for PMCRI, we appended each dataset to itself in either a horizontal or a vertical direction. The base diabetes and yeast datasets each comprise roughly 100,000 data records and 48 attributes. Please note that in these experiments PMCRI's learning algorithm is based on PrismTCS and produces exactly the same rules as the serial version of PrismTCS would induce. Therefore there is no concern with issues relating to the comparative quality of rules generated by different algorithms. As all datasets were based on either the yeast or the diabetes dataset the induced classifiers were identical for all dataset sizes based on yeast and also for all datasets sizes based on diabetes. In particular the classifier induced on yeast produces 467 rules and the classifier on diabetes 110 rules. However, as all datasets comprised 48 attributes we will have imposed a slight workload imbalance for 10 processors as we only assign complete attribute lists to each processor. Thus, for the 10 expert configuration, two expert machines were holding 4 and eight experts were holding 5 attribute lists. The machines we used for all experiments in this section had a CPU speed of 2.8 GHz and a memory of 1 GB. The operating system used was XUbuntu. In general when we talk about processors in this section we in fact mean expert machines.

4.1 Size up And Capability Barriers of PMCRI

In size-up experiments a system's performance is examined on a fixed processor configuration on an increasing workload. In PMCRI the workload is equivalent to the number of data records that are used to train a Prism classifier. In general we hope to achieve a linear size-up meaning that the runtime is a linear function of the data set size.

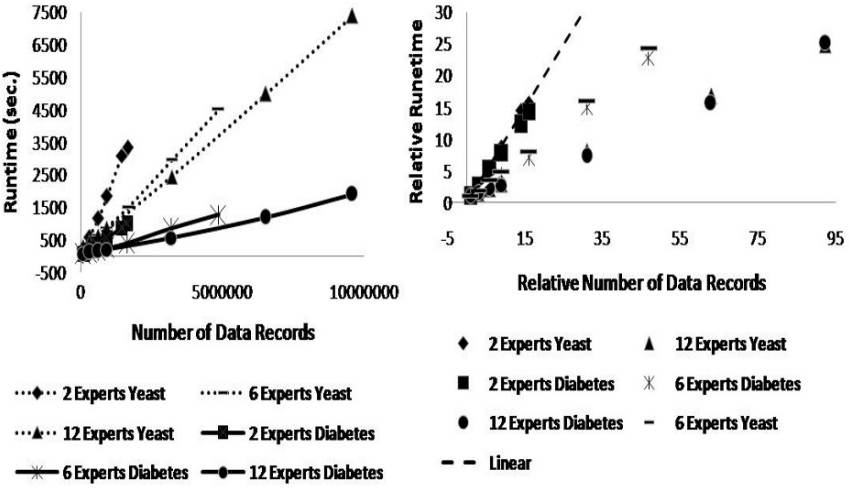

Fig. 4. Size up behaviour of PMCRI on portrait formatted yeast and diabetes dataset

The left hand side of figure 4 shows the runtimes plotted versus the number of yeast and diabetes dataset instances for PMCRI using different numbers of expert machines. In general we observe a linear behaviour for all configurations. What we can also read on the left hand side in figure 4 are the capability barriers of PMCRI. In a particular configuration of PMCRI the capability barrier is equivalent to the amount of data the framework can cope with. If we load too many attribute lists into the memory of the expert machines then the operating system on the expert machines will buffer parts of the lists to the swap memory on the hard drive in order to avoid a memory overflow. This buffering would cause a considerable overhead, thus a too large workload has to be avoided. We can see that the capability barrier of PrismTCS with PMCRI with 2 experts was reached after roughly 166000 data records and for PrismTCS with PMCRI with four machines after roughly 322000 data records. In PMCRI the capability barriers can be widened by adding more machines and thus more memory. Figure 6 shows that for PMCRI if the amount of memory is doubled the capability barriers will also double in size. We took a closer look into the size up behaviour of PMCRI. The right hand side of figure 4 represents the same data as the left hand side with the difference that both axes have been normalised. Now the ideal scaling behaviour

for all processor configurations would be a straight line through the points (1,1), (2,2), (3,3) as displayed in figure 4. We can see that except for the serial version of PrismTCS all data points are below the ideal behaviour and thus indicate a slightly better behaviour than linear for PrismTCS parallelised using PMCRI. This can be explained by the fact that the communication does not increase linearly with the number of data records. There are two types of communication.

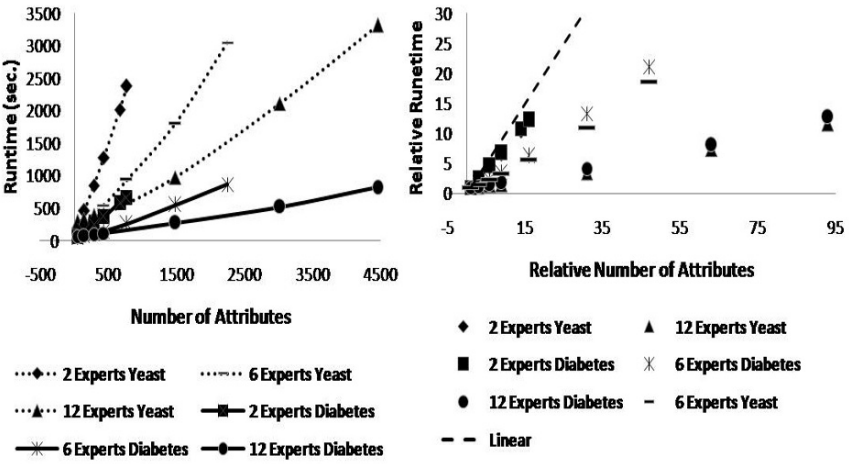

Fig. 5. Size up behaviour of PMCRI on landscape formatted yeast and diabetes dataset

The first type of communication are indices of list records that are covered or uncovered by the currently induced rule term, which increases linearly with the number of data records. However, for each rule term induced, PMCRI will broadcast information about the relevant rule term using the blackboard system. This second type of communication consists of only two values: the covering probability and the count with which the induced rule term covers the target class. This second type of communication is dependent on the total number of rule terms induced and not on the number of data records. As for all sizes of the diabetes dataset we induce the same amount of rule terms, this second type of communication stays constant and is the reason for a size-up being slightly better than linear. Figure 5 illustrates similar experiments as described above for figure 4. The difference is that this time 'landscape' versions of the yeast and diabetes datasets were taken into account. Again we observe a linear behaviour but this time for the runtimes of PMCRI with respect to the number of attributes rather than the number of data instances. We also observe the same behaviour for the capability barriers as for portrait data. We can see that the capability barrier of PrismTCS with PMCRI with 2 experts was reached after 768 attribute lists and for PrismTCS with PMCRI with four machines after roughly 1488 attribute lists. Again we took a closer look into the size-up behaviour of PMCRI. And again we normalised both the axes of the data of the left hand side of figure 5 and plotted

them as shown on the right hand side of figure 5. Once more we observed a size up behaviour better than linear for the same reasons as stated above for the data in portrait format. This time even the first type of communication will stay constant as by adding more attributes the number of indices of list records that are covered or uncovered by the currently induced rule term stays constant as well.

4.2 Speed up of PMCRI

With the speed up factors we compare how much the parallel version of an algorithm is faster using *p* processors compared with one processor.

$$S_p = \frac{R_1}{R_p} \quad (1)$$

Formula 1 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. For example if two processors were used then a speedup factor of 2 means that we gained 100% benefit from using an additional processor. In reality this speedup factor will be below the number of processors for various reasons such as a communication overhead imposed by each processor, which would be in our case caused by communication of information about rule terms and indices of list records. Then there is also the synchronization overhead. For example in the case of PMCRI if a processor has induced the locally best rule term it has to wait for the remaining machines to finish their rule term induction in order to receive or derive the indices that are covered from the globally best rule term. However, as stated in section 3 the relative workload of each processor stays constant thus a synchronisation overhead will not be overwhelming. Figure 6 shows the speed up factors of PrismTCS parallelised using PMCRI for different sizes of diabetes and yeast datasets. The left hand side of figure 6 shows the speedup factors for the data with different numbers of instances and the right hand side of figure 6 for the data with different numbers of attributes. We ran configurations of 2, 4, 6, 8, 10 and 12 processors against the serial version of PrismTCS. We can observe a general tendency that for a fixed dataset size with an increasing number of processors the speedup factors increase until they reach a maximum and then start to decrease again. We can also observe that the larger the dataset size the more processors are needed in order to reach the maximum speedup. Thus in general we can say the larger the amount of training data the more benefit we gain from using more expert machines.

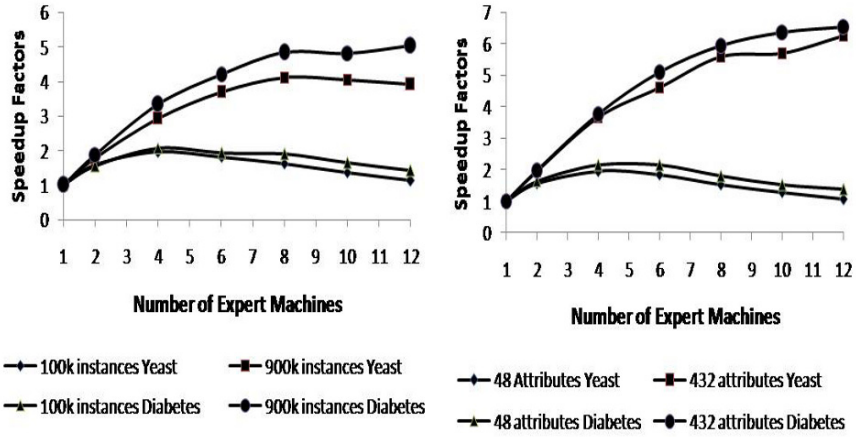

Fig. 6. Speedup behaviour of PMCRI.

5 Conclusions

We presented the work and results of the PMCRI framework, a Parallel Modular Classification Rule Induction framework. PMCRI harvests the computational power of a distributed network in order to make modular classification rule induction scale better on large datasets. We started the paper with a discussion of two general approaches to classification rule induction, the *divide and conquer* and *separate and conquer* approaches. There have been two general approaches to making the classification rule induction approach scale better on large datasets: sampling and parallelisation, but there has been no attempt to parallelise the *separate and conquer* approach. We continued the paper by discussing the Prism family of algorithms that follow the *separate and conquer* approach and discussed its quality and examined its scaling behaviour. We next discussed the suitability of Prism's pre-pruning method, J-pruning, for scaling up algorithms of the Prism family to large datasets. Next we presented the PMCRI framework with an integrated J-Pruning facility that helps to parallelise *separate and conquer* algorithms of the Prism family and similar ones. We evaluated PMCRI experimentally on a parallel implementation of PrismTCS. First we performed size-up experiments in order to determine PMCRI's scaling behaviour with respect to the number of data instances and attributes. In both cases we observed a size-up behaviour better than linear. Next we determined the speedup factors for several numbers of expert machines on several amounts of training data. We observed that the larger the amount of training data regarding the number of attributes and data instances the more processors are needed to achieve the maximum speedup.

References

1. Hunt E. B., Marin J., and Stone P. J., *Experiments in Induction*. 1966: Academic Press.
2. Quinlan J. R., *Induction of decision trees. Machine Learning.* Vol. 1. 1986. 81-106.
3. Michalski R.S., *On* the *quasi-minimal solution of the general covering problem,* in *Proceedings of the Fifth International Symposium on Information Processing.* 1969: Bled, Yugoslavia. p. 125-128.
4. Cendrowska J., *PRISM: an Algorithm for Inducing Modular Rules.* International Journal of Man-Machine Studies, 1987. 27: p. 349-370.
5. Catlett J., *Megainduction: Machine learning on very large databases.* 1991, University of Technology, Sydney.
6. Metha M., Agrawal R., and Rissanen J., *SLIQ: A Fast Scalable Classifier for Data Mining.* International Conference on Extending Database Technology EDBT'96), 1996.
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. Srivastava, A., et al., *Parallel Formulations of Decision-Tree Classification Algorithms.* Data Mining and Knowledge Discovery, 1999. 3(3): p. 237-263.
9. Stahl F., Bramer M., and A. M., *PMCRI: A Parallel Modular Classification Rule Induction Framework., in Sixth International Conference on Machine Learning and Data Mining.* In Press, Springer: Leipzig.
10. 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.
11. Bramer M., *Inducer: a public domain workbench for data mining.* International Journal of Systems Science, 2005. 36(14): p. 909-919.
12. Smyth, P. and R.M. Goodman, *An Information Theoretic Approach to Rule Induction from Databases.* IEEE Trans. on Knowledge and Data Eng, 1991. 4(4): p. 301-316.
13. Blake C. L. and Merz C. J, *UCI repository of machine learning databases.* 1998, University of California, Irvine, Department of Information and Computer Sciences.
14. Stout M., et al., *Prediction of recursive convex hull class assignments for protein residues.* Bioinformatics, 2008. 24(7): p. 916-923.
15. Provost F., *Distributed Data Mining: Scaling up and Beyond,* in *Advances in Distributed and Parallel Knowledge Discovery,* P.C. H. Kargupta, Editor. 2000, AAAI Press / The MIT Press.
16. Nolle L., Wong K. C. P., and Hopgood A., *DARBS: A Distributed Blackboard System.* Twenty-first SGES International Conference on Knowledge Based Systems, 2001.
17. Stahl F. and Bramer M., *P-Prism: A Computationally Efficient Approach to Scaling up Classification Rule Induction, in IFIP International Conference on Artificial Intelligence.* 2008, Springer: Milan.
18. Stahl F. and Bramer M., *Parallel Induction of Modular Classification Rules,* in *Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*. 2008, Springer: Cambridge.