Parallel Random Prism: A Computationally Efficient Ensemble Learner for Classification

Frederic Stahl, David May and Max Bramer

Abstract Generally classifiers tend to overfit if there is noise in the training data or there are missing values. Ensemble learning methods are often used to improve a classifier's classification accuracy. Most ensemble learning approaches aim to improve the classification accuracy of decision trees. However, alternative classifiers to decision trees exist. The recently developed Random Prism ensemble learner for classification aims to improve an alternative classification rule induction approach, the Prism family of algorithms, which addresses some of the limitations of decision trees. However, Random Prism suffers like any ensemble learner from a high computational overhead due to replication of the data and the induction of multiple base classifiers. Hence even modest sized datasets may impose a computational challenge to ensemble learners such as Random Prism. Parallelism is often used to scale up algorithms to deal with large datasets. This paper investigates parallelisation for Random Prism, implements a prototype and evaluates it empirically using a Hadoop computing cluster.

1 Introduction

The basic idea of ensemble classifiers is to build a predictive model by integrating multiple classifiers, so called base classifiers. Ensemble classifiers are often used to

Frederic Stahl
Bournemouth University, School of Design, Engineering & Computing , Poole House, Talbot Campus, Poole, BH12 5BB e-mail: fstahl@bournemouth.ac.uk

David May
University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE e-mail: David.May@myport.ac.uk

Max Bramer
University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE e-mail: Max.Bramer@port.ac.uk

improve the predictive performance of the ensemble model compared with a single classifier [24]. Work on ensemble learning can be traced back at least to the late 1970s, for example the authors of [13] proposed using two or more classifiers on different partitions of the input space, such as different subsets of the feature space. However probably the most prominent ensemble classifier is the Random Forests (RF) classifier [9]. RF is influenced by Ho's Random Decision Forests (RDF) [18] classifier which aims to generalise better on the training data compared with traditional decision trees by inducing multiple trees on randomly selected subsets of the feature space. The performance of RDF has been evaluated empirically [18]. RF combines RDF's approach with Breiman's **b**ootstrap **agg**regat**ing** (Bagging) approach [8]. Bagging aims to increase a classifier's accuracy and stability. A stable classifier experiences a small change and an unstable classifier experiences a major change in the classification if there are small changes in the training data. Recently ensemble classification strategies have been developed for the scoring of credit applicants [19] and for the improvement of the prediction of protein structural classes [30]. Chan and Stolfo's Meta-Learning framework [11, 12] builds multiple heterogeneous classifiers. The classifiers are combined using a further learning process, a *meta-learning algorithm* that uses different combining strategies such as *voting*, *arbitration* and *combining*. Pocket Data Mining, an ensemble classifier for distributed data streams in mobile phone networks has recently been developed [25]. In Pocket Data Mining the base classifiers are trained on different devices in an ad hoc network of smart phones. Pocket Data Mining has been tested on homogeneous and heterogeneous setups of two different data stream classifiers, Hoeffding Trees [15] and incremental Naive Bayes, which are combined using weighted majority voting [27].

Most rule based classifiers can be categorised in the 'divide and conquer' (induction of decision trees) [23, 22] and the 'separate and conquer' approach [28]. 'Divide and conquer' produces classification rules in the intermediate form of a decision tree and 'separate and conquer' produces IF...THEN rules that do not necessarily fit into a decision tree. Due to their popularity most rule-based ensemble base classifiers are based on decision trees. Some ensemble classifiers consider heterogeneous setups of base classifiers such as Meta-Learning [11, 12], however in practice the base classifiers used are different members of the 'divide and conquer' approach. A recently developed ensemble classifier that is inspired by RF and based on the Prism family of algorithms [10, 5, 6] as base classifiers is Random Prism [26]. The Prism family of algorithms follows the 'separate and conquer' approach and produces modular classification rules that do not necessarily fit into a decision tree. Prism algorithms produce a comparable classification accuracy compared with decision trees and in some cases, such as if there is a lot of noise or missing values, even outperform decision trees.

Random Prism has been evaluated empirically in [26] and shows a better classification accuracy in most cases compared with its standalone base classifier. Furthermore unpublished empirical experiments of the authors show that Random Prism has also a higher tolerance to noise compared with its standalone base classifier. Yet some results presented in [26] show that Random Prism consumes substantially more CPU time compared with its standalone Prism base classifier on the same training data size. This is because like many ensemble classifiers such as RF, Random Prism builds multiple bags of the original training data and hence has to process a multiple of the training data compared with its stand alone base Prism classifier. However, as the runtime of ensemble learners is directly dependent to the training data size and the number of base classifiers, they could potentially be parallelised by training the individual classifiers on different processors. Google's MapReduce [14] framework and its free implementation named Hadoop [1] is a software framework for supporting distributed computing tasks on large datasets in a computer cluster. MapReduce's potential to scale up Random Prism is given through ensemble learning approaches that make use of MapReduce such as [21, 29, 3]. However most parallel ensemble approaches are based on decision trees. The work presented in this paper presents a computationally scalable parallel version of the Random Prism ensemble classifier that can be executed using a network of standard workstations utilising Google's MapReduce paradigm. The proposed *Parallel Random Prism* classifier is evaluated empirically using the free Hadoop [1] implementation of the MapReduce framework in a network of computer workstations. Parallelising Random Prism using Hadoop is particularly interesting as Hadoop makes use of commodity hardware and thus is also, from the hardware point of view, an inexpensive solution.

The remainder of this paper is organised as follows: Section 2 highlights the PrismTCS approach, a member of the Prism family of algorithms, which has been used as base classifier. Section 2 also highlights the Random Prism approach. The prototype of Parallel Random Prism is proposed in Section 3 and evaluated empirically in Section 4. Ongoing and possible future work is discussed in Section 5 and concluding remarks are presented in Section 6.

2 Random Prism

This section highlights first the basic rule induction approach: the Prism / PrismTCS classifier and compares the rulesets induced by Prism / PrismTCS with decision trees. The second part of this section introduces the Random Prism algorithm, and discusses its computational performance briefly.

2.1 The PrismTCS Approach

The intermediate representation of classification rules in the form of a decision tree is criticised in Cendrowska's original Prism paper [10]. Prism produces modular rules that do not necessarily have attributes in common such as the two rules below:

IF A = 1 AND B = 1 THEN class = x

IF C = 1 AND D = 1 THEN class = x

These modular rules cannot be represented in a decision tree without adding unnecessary rule terms. For this example it is assumed that that each of the four attributes represented in the rules above have three possible values *1*, *2* and *3*. Further it is assumed that all data instances matching any of the two rules above is labelled with class *x* and the remaining instances with class *y*. According to Cendrowska [10], forcing these rules into a tree structure would lead to the tree illustrated in Figure 1.

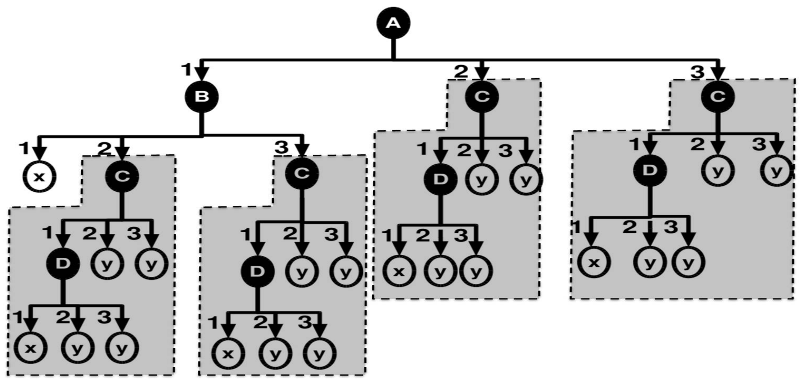

Fig. 1 The replicated subtree problem based on Cendrowska's example in her original Prism Paper. The shaded subtrees highlight the replicated subtrees.

This will result into a large and needlessly complex tree with potentially unnecessary and possibly expensive tests for the user, which is also known as the replicated subtree problem [28].

All 'separate and conquer' algorithms follow the same top level loop. The algorithm induces a rule that explains a part of the training data. Then the algorithm separates the instances that are not covered by the rules induced so far and conquers them by inducing a further rule that again explains a part of the remaining training data. This is done until there are no training instances left [16].

Cendrowska's original Prism algorithm follows this approach. However it does not not scale well on large datasets. A version of Prism that attempts to scale up to larger datasets has been developed by one of the authors [6] and is also utilised for the Random Prism classifier.

There have been several variations of Prism algorithms such as PrismTC, PrismTCS and N-Prism which are implemented in the Inducer data mining workbench [7]. However PrismTCS (Target Class Smallest first) seems to have a better computational performance compared with the other Prism versions whilst maintaining the same level of predictive accuracy [26].

PrismTCS is outlined below using pseudocode [26]. A_x denotes a possible attribute value pair and D the training data. $Rule\_set rules = newRule\_set()$ creates a new ruleset, *Rule rule = new Rule(i)* creates a new rule for class *i*, $rule.addTerm(A_x)$ adds attribute value pair A_x as a new rule term to the rule, and $rules.add(rule)$ adds the newly induced rule to the ruleset.

```
        D′ = D;
        Rule_set rules = new Rule_set();
Step 1: Find class i that has the fewest instances in the training
        set;
        Rule rule = new Rule(i);
Step 2: Calculate for each Ax p(class = i| Ax);
Step 3: Select the Ax with the maximum  p(class = i| Ax);
        rule.addTerm(Ax);
        Delete all instances in D′ that do not cover rule;
Step 4: Repeat 2 to 3 for D′ until D′ only contains instances
        of classification i.
Step 5: rules.add(rule);
        Create a new D′ that comprises all instances of D except
        those that are covered by all rules induced so far;
Step 6: IF (D′ is not empty)
              repeat steps 1 to 6;
```

2.2 Random Prism Classifier

The basic Random Prism architecture is highlighted in Figure 2. The base classifiers are called R-PrismTCS and are based on PrismTCS. The prefix *R* denotes the random component in Random Prism which comprises Ho's [18] random feature subset selection and Breiman's bagging [8]. Both random components have been chosen in order to make Random Prism generalise better on the training data and be more robust if there is noise in the training data. In addition to the random components, J-pruning [6], a rule pre-pruning facility has been implemented. J-pruning aims to maximise the theoretical information content of a rule while it is being induced. J-pruning does that by triggering a premature stop of the induction of further rule terms if the rule's theoretical information content would decrease by adding further rule terms. In general, the reason for chosing PrismTCS as base classifier is that it is the computationally most efficient member of the Prism family of algorithms [26].

In Random Prism the bootstrap sample is taken by randomly selecting *n* instances with replacement from the training data *D*, if *n* is the total number of training instances. On average each R-PrismTCS classifier will be trained on 63.2 % of the original data instances [26]. The remaining data instances, on average 36.8 % of the original data instances, are used as validation data for this particular R-PrismTCS classifier.

The pseudocode below highlights the R-PrismTCS algorithm [26] adapted from the PrismTCS pseudocode in Section 2.1. *M* denotes the number of features in *D*:

```
        D′ = random sample with replacement of size n from D;
        Rule_set rules = new Rule_set();
Step 1: Find class i that has the fewest instances in the training
        set;
        Rule rule = new Rule(i);
Step 2: generate a subset F of the feature space of size m where
        (M>=m>0);
```

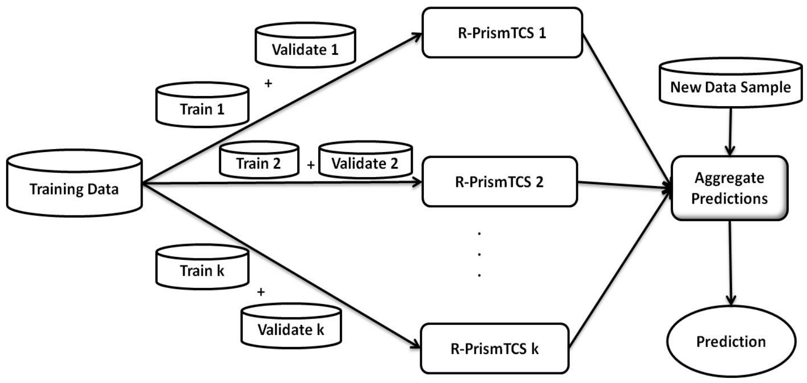

Fig. 2 The Random Prism Architecture using the R-PrismTCS base classifiers and weighted majority voting for the final prediction.

```
Step 3: Calculate for each Ax in F  p(class = i| Ax);
Step 4: Select the Ax with the maximum  p(class = i| Ax);
        rule.addTerm(Ax);
        Delete all instances in D′ that do not cover rule;
Step 5: Repeat 2 to 4 for D′ until D′ only contains instances
        of classification i.
Step 6: rules.add(rule);
        Create a new D′ that comprises all instances of D except
        those that are covered by all rules induced so far;
Step 7: IF (D′ is not empty)
             repeat steps 1 to 7;
```

For the induction of each rule term a different randomly selected subset of the feature space without replacement is drawn. The number of features drawn is a random number between 1 and *M*. The pseudocode outlined below highlights the Random Prism training approach [26], where *k* is the number of base classifiers and *i* is the *ith* R-PrismTCS classifier:

```
double weights[] = new double[k];
Classifiers classifiers = new Classifier[k];
for(int i = 0; i < k; i++)
   Build R-RrismTCS classifier r;
   TestData T = instances of D that have not been used to induce r;
   Apply r to T;
   int correct = number of by r correctly classified instances in T;
   weights[i] = correct/(number of instances in T);
```

The pseudocode highlighted above shows that for each *R-PrismTCS* classifier a weight is also calculated. As mentioned above, for the induction of each classifier only about 63.2% of the total number of training instances are used. The remaining instances, about 36.8% of the total number of instances are used to calculate the classifier's accuracy which we call its weight. As mentioned earlier in this section, Random Prism uses weighted majority voting, where each vote for a classification for a test instance corresponds to the underlying *R-PrismTCS* classifier's weight. This is different from RF and RDF which simply use majority voting. Random Prism also uses a user defined threshold *N* which is the minimum weight a *R-PrismTCS* classifier has to provide in order to take part in the voting.

3 Parallelisation of the Random Prism Classifier

The runtime of Random Prism has been measured and compared with the runtime of PrismTCS as shown in Table 1 and as published in [26]. Intuitively one would expect that the runtime of Random Prism using 100 PrismTCS base classifiers is 100 times slower that PrismTCS. However, the runtimes shown in Table 1 clearly show that this is not the case, which can be explained by the fact that Random Prism base classifiers do not use the entire feature space to generate rules, which in turn limits the search space and thus the runtime. Nevertheless Random Prism is still multiple times slower than PrismTCS, hence parallelisation has been considered in [26] and is now described in this section.

Table 1 Runtime of Random Prism on 100 base classifiers compared with a single PrismTCS classifier in milliseconds.

Dataset	Runtime PrismTCS	Runtime Random Prism
monk1	16	703
monk3	15	640
vote	16	672
genetics	219	26563
contact lenses	16	235
breast cancer	32	1531
soybean	78	5078
australian credit	31	1515
diabetes	16	1953
crx	31	2734
segmentation	234	15735
ecoli	16	734
balance scale	15	1109
car evaluation	16	3750
contraceptive method choice	32	3563

For the parallelisation of Random Prism we used Apache Hadoop, which is an application distribution framework designed to be executed in a computer cluster consisting of a large number of standard workstations. Hadoop uses a technique called *MapReduce*. MapReduce splits an application into smaller parts called *Mappers*. Each Mapper can be processed by any of the workstations in the nodes in the cluster. The results produced by the Mappers are then aggregated and processed by one or more *Reducer* nodes in the cluster. Hadoop also provides its own file system, the Hadoop Distributed File System (HDFS). HDFS distributes the data over the cluster and stores the data redundantly on multiple cluster nodes. This speeds up data access. Failed nodes are automatically recovered by the framework providing high reliability to the application.

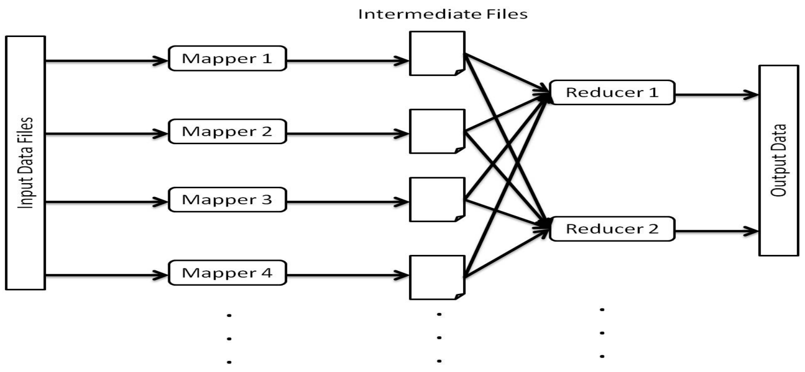

Fig. 3 A typical setup of a Hadoop computing cluster with several Mappers and Reducers. A physical computer in the cluster can host more than one Mapper and Reducer.

Figure 3 highlights a typical setup of a Hadoop computing cluster. Each node in a Hadoop cluster can host several Mappers and Reducers. In Hadoop large amounts of data are analysed by distributing smaller portions of the data to the Mapper machines plus a function to process these data portions. Then the results of the Mappers (intermediate files) are aggregated by passing them to the Reducer. The Reducer uses a user defined function to aggregate them. Hadoop balances the workload as evenly as possible by distributing it as evenly as possible to the Mappers. The user is only required to implement the function for the Mapper and the Reducer.

We utilised Hadoop for the parallelisation of Random Prism as depicted in Figure 4. In Random Prism the Mapper builds a different bagged version of the training data and uses the remaining data as validation data. Furthermore each Mapper gets a R-PrismTCS implementation as a function to process the training and the validation data. The following steps describe the parallelisation of Random Prism using Hadoop:

```
Step 1: Distribute the training data over the computer cluster using the
        Hadoop Distributed File System (HDFS);
Step 2: Start x Mapper jobs, where x is the number of base PrismTCS
        classifiers desired. Each Mapper job comprises, in the following
        order:
         - Build a training and validation set using Bagging;
         - Generate a rulset by training the PrismTCS classifier on
           the training set;
         - Calculated the PrismTCS classifiers weight using the
           validation set;
         - Return the ruleset and the weight.
Step 3: Start the Reducer with a list of rulesets and weights produced
        each each Mapper (PrismTCS classifier);
Step 4: The Reducer retuns the final classifier which is a set of PrismTCS
        rulesets which perform weighted majority voting for each test
        instance.
```

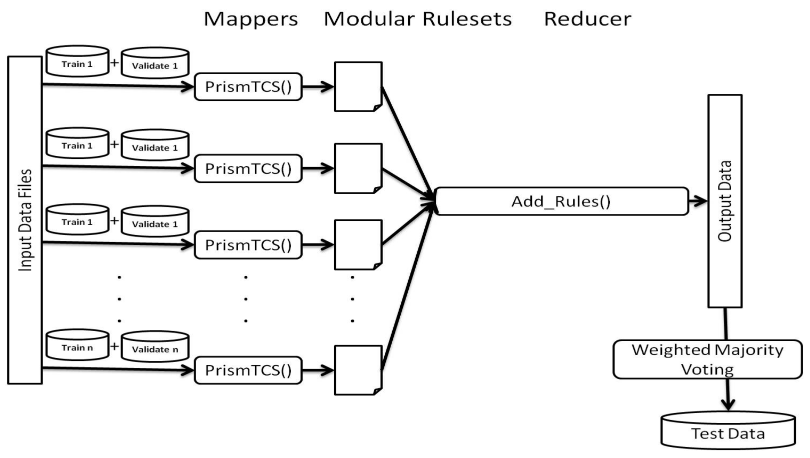

Fig. 4 Parallel Random Prism Architecture

4 Evaluation of Parallel Random Prism Classification

The only difference between the parallel and the serical versions of Random Prism is that in the parallel version the R-PrismTCS classifiers are executed concurrently on multiple processors and in the serial version they are executed sequentially. Hence both algorithms have the same classification performance, but Parallel Random Prism is expected to be faster. For an evaluation of the classification performance of Random Prism the reader is referred to [26]. In this paper we evaluate Parallel Random Prism empirically with respect to its computational performance, using the four datasets outlined in Table 2. The datasets are referred to as tests in this paper. The data for tests 1 to 3 is synthetic biological data taken from the infobiotics data repository [2], and the data for test 4 is taken from the UCI repository [4]. The reason for using the infobiotics repository is that it provides larger datasets compared with the UCI repository. The computer cluster we used to evaluate Parallel Random Prism comprised 10 workstations, each providing a CPU of 2.8 GHz speed and a memory of 1 GB. The operating system installed on all machines is XUbuntu. In this paper the term node or cluster node refers to a workstation and each node hosts two Mappers. As for in the qualitative evaluation in [26], 100 base classifiers were used.

Table 2 Evaluation datasets.

Test	Test Dataset	Number of Data Instances	Number of Attributes	Number of Classes
1	biometric data 1	50000	5	5
2	biometric data 2	15000	19	5
3	biometric data 3	30000	3	2
4	genetics	2551	59	2

Parallel Random Prism has been evaluated in terms of its scalability to different sizes of the training data in ‘size-up’ experiments described in Section 4.1, and in term of its scalability to different numbers of computing nodes in ‘Speed-up’ experiments described in Section 4.2.

4.1 Size-up Behaviour of Parallel Random Prism

The size-up experiments of the Parallel Random Prism system examine the performance of the system on a configuration with a fixed number of nodes/processors and an increasing data workload. In general we hope to achieve a runtime which is a linear function to the training data size. The two datasets with most instances, datasets test 1 and test 3 were chosen for the size-up experiments.

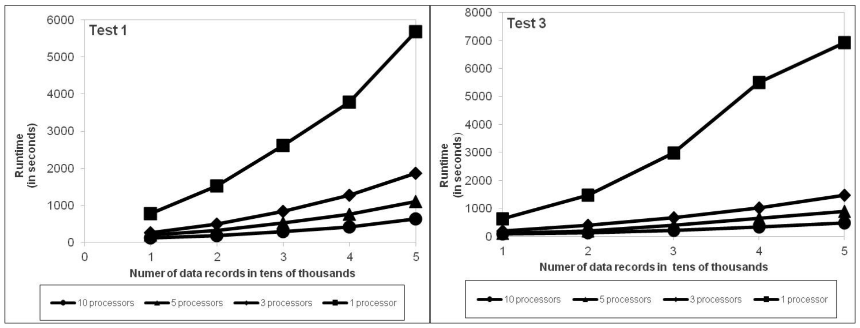

Fig. 5 Size up behaviour of Parallel Random Prism on two test datasets.

For the size-up experiments we took a sample of 10000 data records from the dataset. In order to increase the number of data records we appended the data to itself in the vertical direction. The reason for appending the data sample to itself is that this will not change the concept hidden in the data. If we simply sampled a larger sample from the original data, then the concept may influence the runtime. Thus appending the sample to itself allows us to examine Parallel Random Prism’s performance with respect to the data size more precisely. The calculation of the weight might be influenced by my appending the data to itself as a multiplied data instance may appear in both the test set and the training set. However this is not relevant to the computational performance examined here.

The linear regression equations below are corresponding to the size-up experiments illustrated in Figure 5 and support a linear size up behaviour of Parallel Random Prism on the two chosen datasets, where x is the number of data records/instances in tens of thousands and y the relative runtime in seconds:

$1 \quad processor(test1): \quad y = 1206.8x - 748.8 \quad (R^2 = 0.972)$

$3 \quad processors(test1): \quad y = 396x - 242.6 \quad (R^2 = 0.972)$

$5 \quad processors(test1): \quad y = 226.2x - 99 \quad (R^2 = 0.968)$

10 $processors(test1):\quad y = 128x - 52.8 \quad (R^2 = 0.959)$
1 $processor(test3):\quad y = 1662.2x - 1487 \quad (R^2 = 0.974)$
3 $processors(test3):\quad y = 315x - 193 \quad (R^2 = 0.977)$
5 $processors(test3):\quad y = 197.3x - 143.3 \quad (R^2 = 0.973)$
10 $processors(test3):\quad y = 103.7x - 61.5 \quad (R^2 = 0.965)$
In general we can observe a nice size-up of the system close to being linear.

4.2 Speed-up Behaviour of Parallel Random Prism

A standard metric to measure the efficiency of a parallel algorithm is the speed up factor [17, 20]. The speed up factor measures how much using a parallel version of an algorithm on p processors is faster than using only one processor.

$$S_p = \frac{R_1}{R_p}$$

In this formula S_p represents the speed up factor. R_1 is the runtime of the algorithm on a single machine and R_p is the runtime on p machines. The dataset size stays constant in all processor configurations. In the ideal case the speed up factor would be the same as the number of processors utilised. However, usually the speed up factor will be below the ideal value due to overheads, for example due to the data communication overhead.

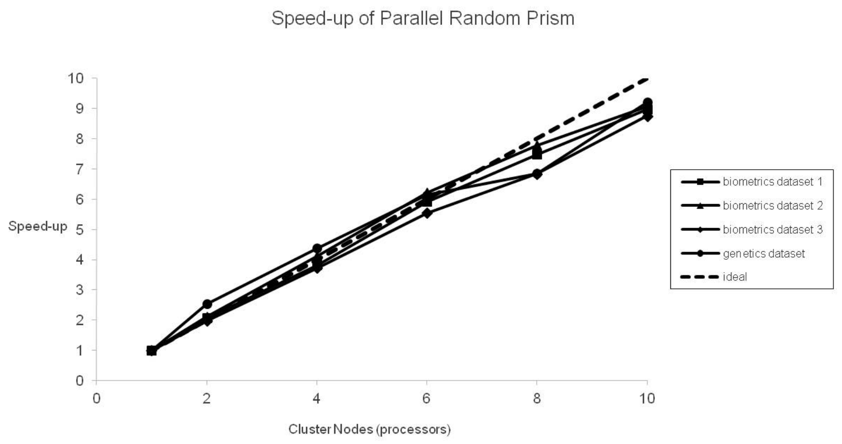

Fig. 6 The Speed-up behaviour of Parallel Random Prism.

Figure 6 shows the speedup factors recorded for Parallel Random Prism on all four test datasets for different numbers of workstations (processors) used in the cluster, ranging from one workstation up to ten, the maximum number of workstations we had available for the experiments. The ideal speed-up is plotted as a dashed line. What can be seen in Figure 6 is that the speedup for all cases is close to the ideal case. However, there is a slightly increasing discrepancy the more workstations are utilised. However this discrepancy can be explained by the increased communication overhead caused by adding more nodes that retrieve bagged samples from the training data distributed in the network. There will be an upper limit of the speedup factors beyond which adding more workstations will decrease the speedup rather than increasing it. However considering the low discrepancy after adding 10 workstations suggests that we are far from reaching the maximum number of workstations that would still be beneficial. Two outliers can be identified for the speed-up factors for the genetics dataset. In fact the speedup is better than the ideal case. Such a 'superlinear' speedup can be explained by the fact that the operating system tends to move frequent operations or frequently used data elements into the cache memory which is faster than the normal system memory. The genetics dataset has many fewer data instances compared with the remaining datasets. Hence a plausible explanation for this superlinear speedup is that distributing the bagged samples between several workstations will result in a larger portion of the total data samples being held in the cache memory. This benefit may outweigh the communication overhead caused by using only two and four workstations. However adding more workstations will increase the communication overhead to a level that outweighs the benefit of using the cache memory.

In general we can observe a nice speedup behaviour suggesting that many more workstations could be used to scale up Parallel Random Prism further.

5 Ongoing and Future Work

The presented evaluation of the scalability of the proposed system was conducted with respect to the number of data instances. However, in some cases the data size might be determined by a large number of attributes, such as in gene expression data. Hence a further set of experiments that examine Parallel Random Prism's behaviour with respect to a changing number of attributes is planned.

A further development of the voting strategy is currently being considered. Depending on the sample drawn from the training data a base classifier may predict different classes with a different accuracy and hence a voting system that uses different weights for different class predictions is currently being developed. In addition to that a further version of Random Prism and thus Parallel Random Prism that makes use of different versions of Prism as base classifier will be developed in the future.

6 Conclusions

This work presents a parallel version of the Random Prism approach. Random Prism's classification accuracy and robustness to noise, has been evaluated in previous work and experiments and shown to improve PrismTCS's (Random Prism's base classifier's) performance. This work addresses Random Prism's scalability to larger datasets. Random Prism does not perform well on large datasets, just like any ensemble classifier that uses bagging. Hence a data parallel approach that distributes the training data over a computer cluster of commodity workstations is investigated. The induction of multiple PrismTCS classifiers on the distributed training data is performed concurrently by employing Google's Map/Reduce paradigm and Hadoop's Distributed File System.

The parallel version of Random Prism has been evaluated in terms of its size-up and speed-up behaviour on several datasets. The size-up behaviour showed a desired almost linear behaviour with respect to an increasing number of training instances. Also the speed-up behaviour showed an almost ideal performance with respect to an increasing number of workstations. Further experiments will be conducted in the future examining Parallel Random Prism's size-up and speed-up behaviour with respect to the number of attributes. In general the proposed parallel version of Random Prism showed excellent scalability with respect to large data volumes and the number of workstations utilised in the computer cluster.

The future work will comprise a more sophisticated majority voting approach taking the base classifier's accuracy for individual classes into account and also the integration of different 'Prism' base classifiers will be investigated.

Acknowledgements The research leading to these results has received funding from the European Commission within the Marie Curie Industry and Academia Partnerships & Pathways (IAPP) programme under grant agreement n^{o} 251617.

References

1. Hadoop, http://hadoop.apache.org/mapreduce/ 2011.
2. Jaume Bacardit and Natalio Krasnogor. The infobiotics PSP benchmarks repository. Technical report, 2008.
3. Justin D. Basilico, M. Arthur Munson, Tamara G. Kolda, Kevin R. Dixon, and W. Philip Kegelmeyer. Comet: A recipe for learning and using large ensembles on massive data. *CoRR*, abs/1103.2068, 2011.
4. C L Blake and C J Merz. UCI repository of machine learning databases. Technical report, University of California, Irvine, Department of Information and Computer Sciences, 1998.
5. M A 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.
6. M A 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.

7. M A Bramer. Inducer: a public domain workbench for data mining. *International Journal of Systems Science*, 36(14):909–919, 2005.
8. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
9. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
10. J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.
11. Philip Chan and Salvatore J Stolfo. Experiments on multistrategy learning by meta learning. In *Proc. Second Intl. Conference on Information and Knowledge Management*, pages 314–323, 1993.
12. Philip Chan and Salvatore J Stolfo. Meta-Learning for multi strategy and parallel learning. In *Proceedings. Second International Workshop on Multistrategy Learning*, pages 150–165, 1993.
13. B.V. Dasarathy and B.V. Sheela. A composite classifier system design: Concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.
14. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.
15. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.
16. J Fuernkranz. Integrative windowing. *Journal of Artificial Intelligence Resarch*, 8:129–164, 1998.
17. John L Hennessy and David A Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, USA, third edition, 2003.
18. Tin Kam Ho. Random decision forests. *Document Analysis and Recognition, International Conference on*, 1:278, 1995.
19. Nan-Chen Hsieh and Lun-Ping Hung. A data driven ensemble classifier for credit scoring analysis. *Expert Systems with Applications*, 37(1):534 – 545, 2010.
20. Kai Hwang and Fay A Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Book Co., international edition, 1987.
21. Biswanath Panda, Joshua S. Herbach, Sugato Basu, and Roberto J. Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proc. VLDB Endow.*, 2:1426–1437, August 2009.
22. Ross J Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
23. Ross J Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
24. Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33:1–39, 2010.
25. F. Stahl, M.M. Gaber, M. Bramer, and P.S. Yu. Pocket data mining: Towards collaborative data mining in mobile computing environments. In *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 2, pages 323 –330, October 2010.
26. Frederic Stahl and Max Bramer. Random Prism: An alternative to random forests. In *Thirty-first SGAI International Conference on Artificial Intelligence*, pages 5–18, Cambridge, England, 2011.
27. Frederic Stahl, Mohamed Gaber, Paul Aldridge, David May, Han Liu, Max Bramer, and Philip Yu. Homogeneous and heterogeneous distributed classification for pocket data mining. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems V*, volume 7100 of *Lecture Notes in Computer Science*, pages 183–205. Springer Berlin / Heidelberg, 2012.
28. Ian H Witten and Frank Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, second edition, 2005.
29. Gongqing Wu, Haiguang Li, Xuegang Hu, Yuanjun Bi, Jing Zhang, and Xindong Wu. Mrec4.5: C4.5 ensemble classification with mapreduce. In *ChinaGrid Annual Conference, 2009. ChinaGrid '09. Fourth*, pages 249 –255, 2009.
30. Jiang Wu, Meng-Long Li, Le-Zheng Yu, and Chao Wang. An ensemble classifier of support vector machines used to predict protein structural classes by fusing auto covariance and pseudo-amino acid composition. *The Protein Journal*, 29:62–67, 2010.