A Scalable Expressive Ensemble Learning using Random Prism: A *MapReduce* Approach

Frederic Stahl[1], David May[2], Hugo Mills[1], Max Bramer[3], and Mohamed Medhat Gaber[4]

[1] University of Reading, School of Systems Engineering, Whiteknights, RG6 6AY Reading, UK,
{f.t.stahl}, {h.r.mills}@reading.ac.uk
[2] Real Time Information Systems Ltd, 1st & 2nd Floors, 8 South Street, Chichester, PO19 1EH, UK,
david.m@rtis.co.uk
[3] University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, Portsmouth, PO1 3HE, UK,
Max.Bramer@port.ac.uk
[4] Robert Gordon University, School of Computing Science and Digital Media, Riverside East Garthdee Road, Aberdeen, AB10 7GJ, UK,
m.gaber1@rgu.ac.uk

Abstract. The induction of classification rules from previously unseen examples is one of the most important data mining tasks in science as well as commercial applications. In order to reduce the influence of noise in the data, ensemble learners are often applied. However, most ensemble learners are based on decision tree classifiers which are affected by noise. The Random Prism classifier has recently been proposed as an alternative to the popular Random Forests classifier, which is based on decision trees. Random Prism is based on the Prism family of algorithms, which is more robust to noise. However, like most ensemble classification approaches, Random Prism also does not scale well on large training data. This paper presents a thorough discussion of Random Prism and a recently proposed parallel version of it called *Parallel Random Prism*. Parallel Random Prism is based on the MapReduce programming paradigm. The paper provides, for the first time, novel theoretical analysis of the proposed technique and in-depth experimental study that show that Parallel Random Prism scales well on a large number of training examples, a large number of data features and a large number of processors. Expressiveness of decision rules that our technique produces makes it a natural choice for *Big Data* applications where informed decision making increases the user's trust in the system.

1 Introduction

Big Data technologies have opened the door wide for researchers to re-engineer their data science products, allowing for unprecedented scalability. Scalability is key to the success of *cloud computing* hosted applications. An enabler approach providing scalability to a wide range of applications is the *MapReduce*

framework [10]. Motivated by the recent developments in this area, we scale up ensemble classification adopting rule-based classifiers, using *MapReduce* framework. Ensemble classification is the training of individual and diverse base classifiers and the integration of their predictive models into a combined classification model. The aim of ensemble classifiers is to increase the predictive accuracy compared with that of a single classifier. One of the best known ensemble learners is Breiman's Random Forests (RF) [7], which is based on Ho's Random Decision Forests (RDF) [14] ensemble classifier and Breiman's **B**ootstrap **agg**regat**ing** (Bagging) approach [6]. Bagging is used in RF to increase the ensemble classifier's stability and accuracy. In unstable classifiers small variations in the training data cause major variations in the classification. The aforementioned ensemble classifiers are based on decision trees. However, alternatives exist, such as Chan and Stolfos *Meta-Learning* [9] which combines heterogeneous classifiers using a *Meta-Learning* that makes use of different classifier combining strategies such as *voting*, *arbitration* and *combining*.

Most rule based classifiers are either based on the 'divide and conquer' or the 'separate and conquer' rule induction approaches [24]. 'Divide and conquer' based classifiers produce a decision tree, such as Quinlan's C4.5 decision tree induction algorithm [18]; 'separate and conquer' based classifiers produce a set of IF...THEN classification rules, such as the Prism family of algorithms [8, 4, 5]. As pointed out in [20], most ensemble classifiers are based on the 'divide and conquer' approach even though Prism classifiers have shown to be less vulnerable to overfitting compared with decision tree based classifiers [4]. This is especially the case when confronted with noise and missing values in the data [4]. A recently developed ensemble classifier named *Random Prism* [20], which is inspired by RF and RDF, makes use of the Prism family of algorithms as base classifiers. An empirical evaluation of the Random Prism classifier shows that it outperforms its standalone base classifier in terms of a better classification accuracy [20]. Further empirical experiments [20] show that Random Prism also has a higher tolerance to noise compared with its base classifier.

However, also pointed out in [20] Random Prism's CPU time consumption is also considerably higher compared with that of a standalone Prism classifier. This is because Random Prism builds for each base classifier a bag of size N of the original training data [22], if N is the number of data instances in the original training data. Thus even modest sized training data impose a considerabel computational challgenge to ensemble learners using bagging, such as Random Prism. A bag is a collection of data instances in which each data instance may occur more than once. In order to tackle this problem of scalability to larger data a parallel version of the *Random Prism* classifier, called *Parallel Random Prism*, has been developed [22]. Parallel Random Prism is based on data parallelisation and makes use of Google's MapReduce programming paradigm [10]. In particular, Parallel Random Prism uses the Hadoop implementation of MapReduce in order to distribute the induction of each individual base classifier on its own bag to different machines in a computer cluster [1]. Thus the base classifiers are induced concurrently. In this paper we use the expression *parallel* and *distributed*

in the context of algorithms interchangeably, both referring to the concurrent exection of base classifiers through distribution of the training data to multiple computer cluster nodes.

This paper provides a detailed and exhaustive description of Random Prism and Parallel Random Prism approaches. Additionally, it also provides, for the first time, a formal theoretical scalability analysis of Random Prism and Parallel Random Prism, which examines the scalability to much larger computer clusters. This contribution provides a theoretical underpinning that can be used for scalability of the *MapReduce* framework. It also presents a thorough experimental study of Parallel Random Prism's scalability. In particular we look into its scalability with respect to the number of training examples and number of features. It is worth noting that we use the terms 'feature' and 'attribute' interchangeably in this paper.

This paper's structure is as follows: Section 2 presents the Random Prism ensemble learner. The parallel version of Random Prism is outlined in Section 3. Section 4 provides a theoretical scalability analysis of a standalone Prism classifier, the Random Prism ensemble learner and then the Parallel Random Prism approach. This formal scalability analysis is then supported by an empirical evaluation in Section 5. Finally, Section 6 closes the paper with some concluding remarks.

2 Random Prism

As aforementioned Random Prism is inspired by RDF and RF. Ho's RDF approach induces multiple trees, each induced on a random subset of the feature space [14]. This is done in order to make the individual trees generalise better on the training data, which Ho evaluated empirically. RF similarly to RDF induces the trees on feature subsets. However, differently from RDF, RF uses a new random subset of the feature space for evaluating the possible splits of each node in the decision tree [7]. In addition, RF also uses 'Bagging' [6] in order to further increase the predictive accuracy of the ensemble classifier. This is according to [12] because the composite classifier model reduces the variance of the individual classifiers. However, the authors of [11] suggest that bagging not necessarily always reduces variance, but also equalises the influence of training examples and thus stabilises the classifier. Bagging builds for each base classifier a bootstrap sample D_i of a training dataset D using sampling with replacement [6]. Most commonly D_i is of size N where N is the number of training instances in D.

In this paper, we adopt PrismTCS which is a computationally efficient member of the Prism family, and also maintains a similar predictive accuracy compared with the original Prism classifier [5]. A good computational efficiency is needed as ensemble learners generally do not scale well to large datasets. Due to bagging even modest sized training data present a considerable computational challgenge to ensemble learners. In addition, the implemented base classifier makes use of J-pruning as it not only generalises the induced classifier further, but also lowers its runtime [19]. This is because J-pruning will reduce the number

of rule terms induced and thus lower the number of iterations of the base classifier [19]. The random feature subset selection of a random size is also implemented inside the base classifier. This takes place for each rule and for each term expansion of that rule. The resulting base classifier has been termed 'R-PrismTCS', where the 'R' stands for the 'Random' components in the base classifier (random feature subset selection for each rule term and bagging).

Algorithm 1 shows the steps of R-PrismTCS with the exception of J-pruning. F denotes the total number of features, D is the original training data and $rule\_set$ is an initially empty set of classification rules. The operation $rule.addTerm(A_x)$ adds attribute value pair A_x as a rule term to $rule$ and the operation $rule\_set.add(rule)$ adds $rule$ to $rule\_set$. In step 2 for each A_x the conditional probability $p(class = i|A_x)$ is calculated, which is the probability with which A_x covers the target class i.

Algorithm 1: R-PrismTCS Algorithm

D' = build random sample with replacement from D;
$D'' = D'$;
Step 1: find class i that has the fewest instances in D'';
rule = new empty rule for target class i;

Step 2: generate a feature subset f of size m, where $(F > m > 0)$;
calculate for each A_x in f $p(class = i|A_x)$;

Step 3: select the A_x with the maximum $p(class = i|A_x)$;
$rule.addTerm(A_x)$;
delete all instances in D'' that do not cover rule;

Step 4: repeat 2 to 3 for D'' until D'' only contains instances of target class i;

Step 5: $rule\_set.add(rule)$;
create a new D'' that comprises all instances of D' except those that are covered by all rules induced so far;

Step 6: IF (number of instances D'' >1){ repeat steps 1 to 6 };

Figure 1 shows the conceptual architecture of Random Prism. Each R-PrismTCS base classifier is induced on a training sample of size N from the training data, where N is also the size of the training data. This sample is drawn using random sampling with replacement. This statistically results in samples that contain 63.2% of the original instances, some of them drawn multiple times. The remaining 36.8% of the instances that have not been drawn are used as validation data to estimate the individual R-PrismTCS classifier's predictive accuracy ranging from 0 to 1. We call this accuracy the classifier's weight. The individual classifier's weights are then used to perform weighted majority voting on unlabelled data instances. The weights can also be used to filter base classifiers, i.e., retain the classifiers with high predictive accuracy and eliminate those with a poor one according to a user's predefined threshold.

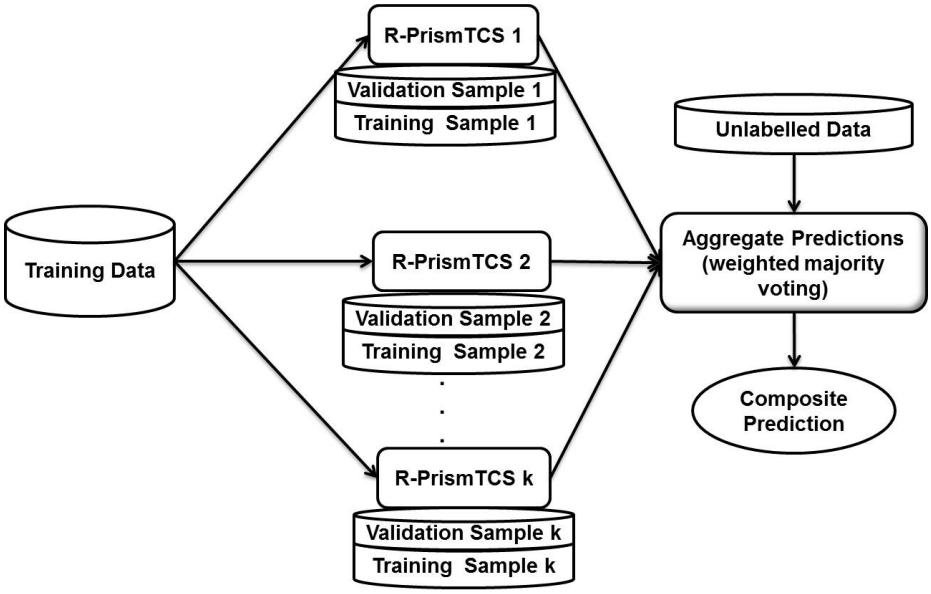

Fig. 1. The architecture of the Random Prism ensemble classifier.

Random Prism's predictive accuracy has been evaluated empirically on several datasets of the UCI repository [3, 20]; and it has been found that Random Prism's classification accuracy is superior to that of RrismTCS's. Furthermore results published recently in [20], show that Random Prism's potential unfolds when there is noise in the training as well as in the test data. Here Random Prism clearly outperforms PrismTCS [20].

However, this paper is more concerned with the scalability of Random Prism to large datasets. One would expect that the runtime of Random Prism inducing 100 base classifiers is approximately 100 times longer, compared with PrismTCS, as Random Prism induces base classifiers with a bag of size N for each base classifier, where N is the total number of training instances. Yet, this is not the case according to the results published in [20]. The reason for this is the random component in R-PrismTCS, which only considers a random subset of the total feature space for the induction of each rule term. Thus the workload of each R-PrismTCS classifier for evaluating candidate features for rule term generation is reduced by the number of features not considered for each induced rule term.

3 The Parallel Random Prism Classifier

This section addresses our proposal to scale up Random Prism ensemble learner by introducing a parallel version of the algorithm. This will help to address the increased CPU time requirements, and also the increased memory requirements. The increased memory requirements are due to the fact that there are k data

samples of size N required if k is the number of R-PrismTCS classifiers and N the number of total instances in the original training data. If k is 100, then the required memory would be 100 times larger compared with the memory requirements of the standalone PrismTCS classifier. The CPU requirements of Random Prism are high, but not 100 times higher due to the random feature subset selection. The parallelisation of the algorithm allows harvesting of the memory and CPU time of multiple workstations for inducing the Random Prism ensemble classifier.

In *data parallelism* smaller portions of the data are distributed to different computing nodes on which data mining tasks are executed concurrently [23]. Ensemble learning lends itself to data parallelism as it is composed of many different data mining tasks, the induction of base classifiers, which can be executed independently, and thus concurrently. Hence, a data parallel approach has been chosen for Random Prism. However, there are some limiting factors concerning scalability, which will be analysed in Section 4.

Section 3.1 highlights the MapReduce paradigm which has been adopted for the parallelisation of Random Prism, and Section 3.2 highlights the architecture of Parallel Random Prism.

3.1 Parallelisation Using the MapReduce Paradigm

A programming paradigm for parallel processing introduced by *Google* is *MapReduce* [10]. It provides a simple way of developing 'data' parallel data mining techniques and thus lends itself to the parallel development of ensemble learners [17]. In addition, MapReduce computer cluster implementations, such as the open source *Hadoop* implementation [1] provide fault tolerance and automatic workload balancing. Hadoop's MapReduce implementation is based on the Hadoop Distributed File System (HDFS), which distributes the data over the computer cluster and stores it redundantly in order to speed up the data access and establish fault tolerance.

Figure 2 illustrates a Hadoop computer cluster. MapReduce partitions an application into smaller parts implemented as *Mapper* components. Mappers can be processed by any computing node within a MapReduce cluster. The aggregation of the results produced by the Mappers is implemented in one or more *Reducer* components, which again can be processed by any computing node within a MapReduce cluster.

MapReduce's significance in the area of data mining is evident through its adoption for many data mining tasks and projects in science as well as in businesses. For example, by 2008 Google made use of MapReduce in over 900 projects [10], such as clustering of images for identifying duplicates [16]. In 2009 the authors of [17] used MapReduce in order to induce and assemble numerous ensemble trees in parallel.

Random Prism can be broken down into multiple R-PrismTCS classifiers induced on bagged samples of the training data. Loosely speaking, Random Prism can be parallelised using Hadoop through implementing R-PrismTCS classifiers as Mappers which can be executed concurrently in a MapReduce cluster. More

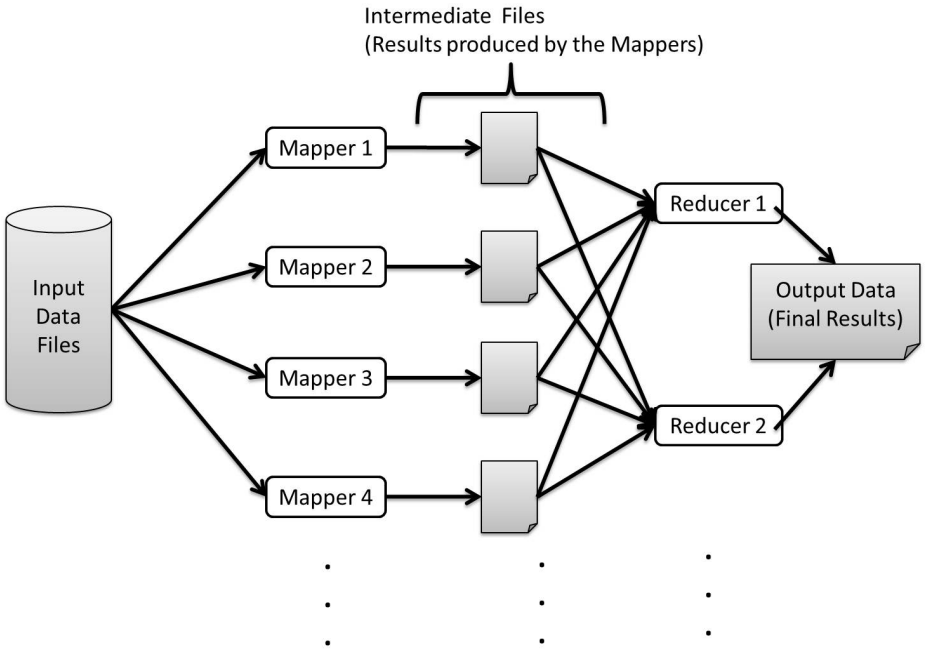

Fig. 2. A typical setup of a Hadoop computing cluster. A physical node in the computer cluster can execute more than one Mapper and Reducer.

details on the Parallel Random Prism architecture are highlighted next in Section 3.2.

3.2 Parallel Random Prism Classifier

Several aspects of the Random Prism algorithm have to be considered for the parallelisation through data parallelism with the MapReduce paradigm. These are the bagging procedure, the induction of R-PrismTCS classifiers and the combination of the individual classifiers into a composite classifier.

Induction of R-PrismTCS Classifiers As mentioned in Section 3.1, Random Prism can be broken down into multiple R-PrismTCS classifiers induced on bagged samples of the training data. These R-PrismTCS classifiers can be induced independently. The only operation that requires the input of all classifiers is the aggregation of their individual sets of classification rules and their weights. Hence, the induction of a R-PrismTCS classifier is implemented directly in a Mapper. Multiple instances of this Mapper can be executed concurrently in a Hadoop cluster. If there are more instances of Mappers than computing nodes, then several Mappers queue to be executed on a node. Thus we keep

the comupational nodes utilised through pilelining. However, the execution of p Mappers at the same time is still concurrent, where p is the number of available computing nodes in the cluster. Once the last mappers are executed on the cluster there may be a small synchronisation overhead as some mappers may finish earlier than others, thus leaving some of the computational nodes idle, but only in the very last stage of the algorithm's execution.

Bagging Procedure The building of a boot strap sample from the training data, using bagging, needs to be executed for each R-PrismTCS classifier in order to create as diverse samples as possible (as required by Random Prism). Thus bagging imposes a considerable computational overhead, which needs to be addressed as well. In the proposed Parallel Random Prism classifier implementation, multiple bagging procedures are executed concurrently. This is realised by integrating the bagging procedure in the Mapper that implements R-PrismTCS. Thus the execution of p bagging procedures at the same time is concurrent, if p is the number of available computing nodes in the cluster. The original training is distributed to each computing node in the Hadoop cluster at the beginning of Parallel Random Prism's execution. We have not taken influence on how Hadoop distributes the data. However, Hadoop typically distributes chunks and redundant copies of the training data across the cluster. This partition and redundancy reduces the communication overhead as well as provides more robustness in the case a cluster node fails. This is done in order to keep the communication overhead low. This way the original training data only needs to be communicated once, as the local Mappers on a computing node only need the local copy of the training data in order to build their individual samples.

Building of Composite Classifier The aggregation of the individual R-PrismTCS classifiers and their associated weights is implemented in a single Reducer. Once the individual R-PrismTCS Mappers finish the induction of their rulesets, they send the rulesets and their associated weights to the Reducer. The Reducer simply holds a collection of classifiers with the weight. If a new unlabelled data instance is presented, then the Reducer applies a weighted majority voting of each classifier, or a subset of the best classifiers (according to their weight), in order to label the new data instance. The data that is transmitted from the Mapper to the Reducer is relatively small in size comprising all the rules of the induced R-PrismTCS base classifiers. Nevertheless, we have incorporated this communication in our analysis in Section 4. However, assuming that the number of R-PrismTCS classifiers is increasing, one may consider distributing the computational and communication overhead (associated with the aggregation of the classifiers) over several Reducers executed on different computational nodes.

Parallel Random Prism Architecture Figure 3 shows the principal architecture of Parallel Random Prism using four Mappers, one Reducer and three cluster nodes.

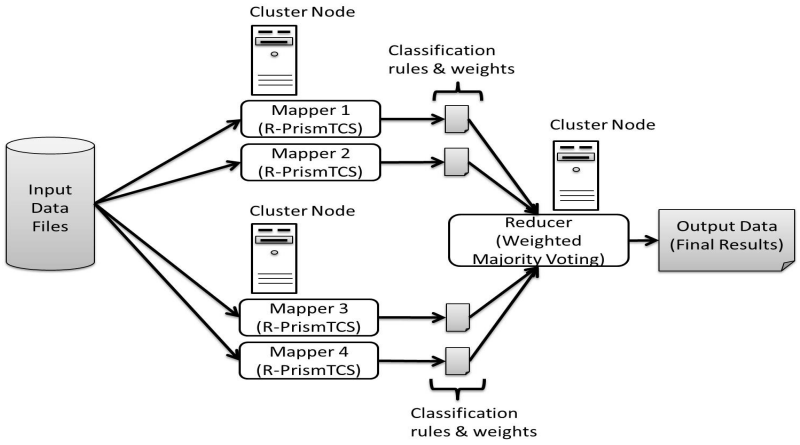

Fig. 3. The Parallel Random Prism Architecture on a Hadoop cluster with two computational nodes, four Mappers and one Reducer.

The input data (training data) is sent to each computing node. A computing node can execute multiple Mappers. Each Mapper implements the R-PrismTCS base classifier outlined in Algorithm 1, creates a validation and a training set and then produces a set of rules using the training data and a weight using the validation data. Then each R-PrismTCS Mapper sends its ruleset and the associated weight (determined using the validation data) to the Reducer. The Reducer keeps a collection of the received classifiers and their weights and applies a weighted majority voting of each, or a subset of the best classifiers, to new unlabelled data instances. The basic steps of Parallel Random Prism are outlined in Algorithm 2.

Algorithm 2: Parallel Random Prism Algorithm

Step 1: Distribute a copy of the training data to each node in the cluster using the Hadoop Distributed File System;

Step 2: Start k Mappers, where k is the number of R-PrismTCS classifiers desired. Each Mapper comprises, in the following order;
- Build a training and validation set using Bagging;
- Induce a ruleset by applying R-PrismTCS on the training data;
- Calculate the ruleset's weight using the validation data;
- Send the ruleset and its weight to the Reducer;

Step 3: Optionally the Reducer applies a filter to eliminate the worse and retrain the strongest rulesets according to their weights;

Step 4: The Reducer returns the final classifier, which is a set of R-PrismTCS rulesets, which perform weighted majority voting for each new unlabelled data instance;

4 Theoretical Analysis of Parallel Random Prism

The complexity of PrismTCS is based on the number of probability calculations for possible split values. In this paper this is denoted as the number of cutpoints. In the ideal case, there would be one feature that perfectly separates all the classes, or simply all data instances would belong to the same class. An average case is difficult to estimate, as the number of iterations of PrismTCS is dependent on the number of rules and rule terms induced, which in turn are dependent on the concept encoded in the training data. However, it is possible to estimate the worst case, assuming that N is the number of instances and M the number of features in the training data. Furthermore a categorical feature will occur at most in only one term per rule, whereas a continuous feature may occur in two terms per rule, as two rule terms can describe any value interval in a continuous feature. Thus in the worst case all features are continuous and all rules have $2\dot{M}$ terms. Also in the worst case each (with the exception of 1) instance is encoded in a separate rule which will lead to $N-1$ rules in total. The -1 is because if there is only one instance left in step 6 of the PrismTCS pseudocode, then there is no need to generate a further rule for it. The complexity (number of cutpoint calculations) of inducing the r^{th} rule is $2M(N-r)$. The factor $(N-r)$ is the number of training instances not covered by the rules induced so far, as mentioned above, in the worst case each rule covers only one training instance. These uncovered instances are used for the induction of the next rule. For example, the number of cutpoint calculations for a term of the first rule ($r=1$), where the training data is still of size N, would be $2M(N-1)$. The total number of cutpoint calculations for the whole rule in this case $(r-1)$ would be $2M(N-1)$ as there are $2M$ rule terms. This summed up for the whole number of rules leads to:

$$T_{PrismTCS} = \sum_{r=1}^{N-1} (2M) \cdot (N-r) = 2M \cdot \frac{N \cdot (N-1)}{2}$$

Which is equivalent to a complexity of $O(N^2 \cdot M)$. Please note that this estimate for the worst case is very pessimistic and unlikely to happen. In reality larger datasets often contain many fewer rules than there are data instances [21]. This is because Random Prism is a stable classifier due to the usage of R-PrimTCS as base classifier and bagging. Also stated before in Section 2, Random Prism employs J-pruning which further reduces the number of rule terms per rule [19]. Hence, in this case linearity can be exhibited as N^2 is reduced to R^2 where R is the total number of rules. An empirical study presented in [21] suggests that the number of rules and rule terms induced does not increase linearly with the number of instances. Also results in [19] suggest a more linear scalability of PrismTCS.

Assuming on average a linear complexity $O(N \cdot M)$ for PrismTCS, the complexity with respect to N and M of Random Prism is a product of four factors. The factors are PismTCS's complexity $O(N \cdot M)$, the average percentage of features f considered by R-PrismTCS (this is a diminishing factor ranging between

0 and 1), the number of classifiers b (which is an increasing factor of a whole number of at least 1 or higher) and a further diminishing factor d which reflects the decrease of rules caused by having repeated instances in the training data for each R-PrismTCS classifier. This leads to $O(N \cdot M) \cdot f \cdot b \cdot d$. As pointed out in Section 2, one would intuitively expect the runtime of Random Prism to be 100 times longer, assuming 100 base classifiers are induced, compared with the serial PrismTCS classifier. Yet the results in [20] show that the runtimes are longer but not 100 times longer. In this particular case the increasing factor b would be 100. However, factors f and d are diminishing and thus have a shortening influence on the runtime. In general as none of these factors comprises an increasing dependence on N or M, this can be approximated to an overall linear complexity of $O(N \cdot M)$. Please note that the complexity of building the composite classifier is not dependent on the training data size but on the number of classifiers. Also building the composite classifier is a computationally relatively inexpensive operation. The bagging is also of linear complexity $O(N)$ assuming that the bag is of size N, as in Random Prism.

Stepping away from the complexity, the actual runtime T_{total}, which is needed to execute the serial version of Random Prism, can be described by:

$$T_{total} = \sum_{i=1}^{b} (T_{sam,i} + T_{cla,i} + T_{asm,i})$$

where T_{total} is the total serial runtime, b is the number of base classifiers, $T_{sam,i}$ is the time needed for sampling (using bagging) for classifier i; $T_{cla,i}$ is the execution time for classifier i and $T_{asm,i}$ is the time needed to integrate classifier $i's$ ruleset into the composite classifier. This description of T_{total} will be used as a base for describing Parallel Random Prism's runtime requirements.

As discussed, the basic Random Prism total runtime description T_{total} can be extended for describing the Parallel Random Prism runtime as shown in the equation below, where p is the number of computing nodes in the Hadoop cluster:

$$T_{total}(p) = T_{comdat} \cdot p + \frac{\sum_{i=1}^{b} T_{sam,i}}{p} + \frac{\sum_{i=1}^{b} T_{cla,i}}{p} + \frac{\sum_{i=1}^{b} T_{comres,i}}{p} + \frac{\sum_{i=1}^{b} T_{asm,i}}{r}$$

$T_{comdat} \cdot p$ is a new term that describes the time needed to communicate the training data to p mappers. p is defined as $p = n \cdot \delta$, where n is the number of computational nodes int the cluster and δ is the number of mappers hosted per n. T_{comres} is also a new term that describes the time needed to communicate the R-PrismTCS rulesets and weights to the Reducer. $T_{sam,i}$, $T_{cla,i}$ and $T_{asm,i}$ are the same terms as in the equation for the serial Random Prism algorithm. However, in the parallel version $T_{sam,i}$ (sampling using bagging) and $T_{cla,i}$ (R-PrismTCS induction) are executed concurrently using multiple Mappers on p processors and hence their runtime can be divided by p.

$T_{asm,i}$ (assembling of the composite classifier) is executed on r Reducers in the Hadoop cluster. Hence the division by r. However, in the setup used for

the experiments in Section 5 only one reducer has been used, hence $r = 1$ in the empirical results. The reason for setting $r = 1$ is because the computational requirement for $T_{asm,i}$ is very low. The only two terms that are not parallelised are $T_{comdat} \cdot p$ and $T_{asm,i}$ and thus these present a computational bottleneck. However, for term $T_{comdat} \cdot p$, the data transmitted to each node is a copy of the original data and it is assumed that the time needed to perform the transmission to each node is the same. Further assume that a star topology network is used with a switch in the centre node, which is the actual setup we used for our empirical evaluation in Section 5. In this case a multicast can be used which transmits the training data from the original node only once to the switch, which then multiplies the data and distributes them to each computing node on a separate wire. Hence, in this case, we can ignore the multiplication of $T_{comdat,i}$ with p as in this case $p = 1$. $T_{asm,i}$ remains a computational bottleneck, which increases with the number of base classifiers. However, its computational requirements are relatively low even for large numbers of base classifiers and is not expected to have a large impact on $T_{total}(p)$. Nevertheless, it would be possible to parallelise $T_{asm,i}$, at least to a certain extent, by using multiple Reducers executed on different cluster nodes. One Reducer per two Mappers could combine the rule sets of these two mappers. The Reducers' outputs (again rules sets) could then be combined similarly using further Reducers executed on different cluster nodes. This may be beneficial for very large numbers of base classifiers. The speed-up factor is a standard metric to evaluate the scalability of parallel algorithms with respect to the number of computing nodes or processors p being used [13, 15]. It shows how much a parallel version of an algorithm is faster compared with its single processor version. The generic formula for the speed-up $S(p)$ is:

$$S(p) = \frac{runtime\ T\ on\ 1\ processor}{runtime\ T\ on\ p\ processors}$$

For Parallel Random Prism the numerator of $S(p)$ can be substituted by $T_{total}(1)$ and the denominator of $S(p)$ can be substituted by $T_{total}(p)$. Thus the speed-up for Parallel Random Prism can be described by:

$$S(p) = \frac{T_{total}(1)}{T_{total}(p)} = \frac{T_{comdat} + \sum_{i=1}^{b} T_{sam,i} + \sum_{i=1}^{b} T_{cla,i} + \sum_{i=1}^{b} T_{comres,i} + \sum_{i=1}^{b} T_{asm,i}}{T_{comdat} \cdot p + \frac{\sum_{i=1}^{b} T_{sam,i}}{p} + \frac{\sum_{i=1}^{b} T_{cla,i}}{p} + \frac{\sum_{i=1}^{b} T_{comres,i}}{p} + \frac{\sum_{i=1}^{b} T_{asm,i}}{r}}$$

Again, what can be seen is that the only limiting factors are $T_{comdat} \cdot p$ and $\sum_{i=1}^{b} T_{asm,i}$ in the denominator of $S(p)$ as they are not parallelised. Yet, the time needed to execute $T_{asm,i}$ and $T_{comdat} \cdot p$ is neglectably small compared with the parallelised portions of Parallel Random Prism. Thus we can assume that the $S(p)$ will be close to the ideal case, which is $S(p) = p$. For example, if running Parallel Random Prism consumes $10000ms$ on one node, then for using

4 nodes we would expect the runtime to be $2500ms$ (4 times faster assuming the ideal case), hence $S(4) = \frac{10000ms}{2500ms} = 4$.

The formula for $S(p)$ above could also be used to determine the theoretical maximum speed-up, through building the derivative $S'(p)$, and calculating its x-axis intercepts and then determining subsequently its global maxima. However, we refrain from this step.

Next Section 5 will provide an empirical analysis of Parallel Random Prism supporting the theoretical analysis presented in this section.

5 Empirical Scalability Study

The empirical study comprises size-up and speed-up experiments on several benchmark datasets. Size-up experiments examine the algorithm's performance (runtime) with respect to the size of the training data; and speed-up experiments examine the algorithm's performance with respect to the number of computing nodes used, using speed-up factors as highlighted in the previous section. For the experiments we used two synthetic datasets from the infobiotics data repository [2]. We have chosen these datasets as they can still be run on a single computing node in our cluster, which can be used as a reference point. The datasets are outlined in Table 1. The Hadoop cluster is hosted on 10 identical off the shelf workstations, each comprising 1 GB memory, 2.8 GHz CPUs and a XUbuntu operating system. The Hadoop version installed on the cluster was 0.20.203.0rc1. All experiments highlighted in this section measure the total runtime from the loading of the data to the cluster, up to aggregating the results at the Reducer.

Table 1. Datasets used for evaluation. Attributes hold double values and class values are represented by a single character.

Test Data	Number of Data Instances	Number of Attributes	Number of Classes
1	50000	5	5
2	30000	3	2

Again, size-up experiments examine the performance of Parallel Random Prism on a fixed number of cluster nodes with an increasing workload (training data size). In general a linear increase in the runtime with respect to the training data size is desired. We produced larger versions of the two datasets in Table 1 by appending the data to itself in vertical (multiplying instances) and horizontal directions (multiplying attributes). Please note that this appending of data does not introduce new concepts and hence does not take influence on the rulesets produced. This is important as altered rule sets may result in different runtimes of the system, and hence the size-up comparison would not be reliable.

The reasoning for this way of increasing the data size is that it will not change the concept encoded in the data. Simply taking different sized samples

from the original training data will influence the concept and thus the runtime needed to find rules describing the concept. Appending the data to itself allows Parallel Random Prism's runtime to be examined more precisely. The calculation of the weight of the individual R-PrismTCS classifiers might be influenced by this way of building different sized samples as some instances may appear in both, the training and the test set. However, this is not relevant for these experiments, as this evaluation examines the computational performance and not the classification accuracy. For all experiments we used 100 R-PrismTCS base classifiers.

The first set of size-up experiments looks on the algorithm's performance with respect to the number of data instances. For each dataset an initial sample of 10000 instances has been taken. Then this sample has been appended to itself in a vertical direction as explained above. The runtime for different sizes of data has been recorded and is plotted in Figure 4 versus the data size. Please note that an initial sample of 10000 instances may seem small. However, considering the usage of 100 base classifiers would increase the sample in the memory so that the Parallel Random Prism system has in fact to deal with 1000000 data instances for a 10000 instance input sample.

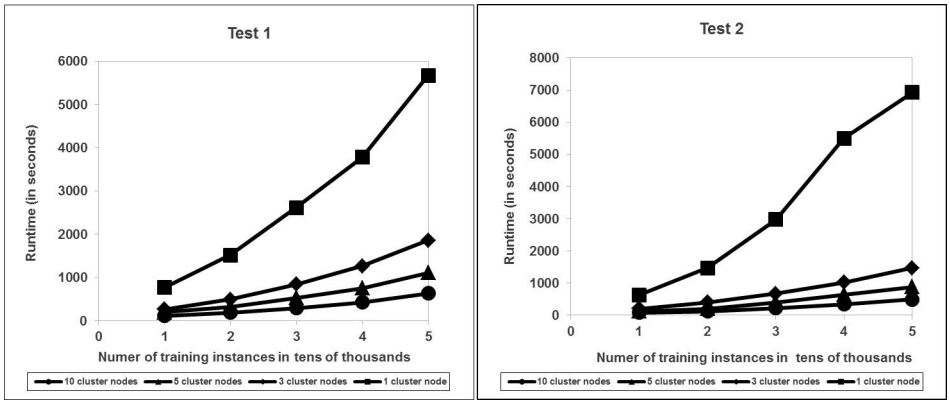

Fig. 4. Size up behaviour of Parallel Random Prism with respect to the number of training instances. Headings Test 1 and Test 2 refer to the test datasets in Table 1. These datasets have in this case been appended to themselves in order to increase the number of training instances, while keeping the concept stable.

In general we can observe a nice size-up that is close to being linear with respect to the number of training instances. These results clearly support the theoretical average linear behaviour.

The second set of size-up experiments looks at the algorithm's performance with respect to the number of features. The data has been appended to itself in a horizontal direction as explained earlier in this section. Again, the number of training instances is increasing by factor 100 due to the use of 100 base classifiers.

The runtime for different sizes of data has been recorded and is plotted in Figure 5 versus the data size.

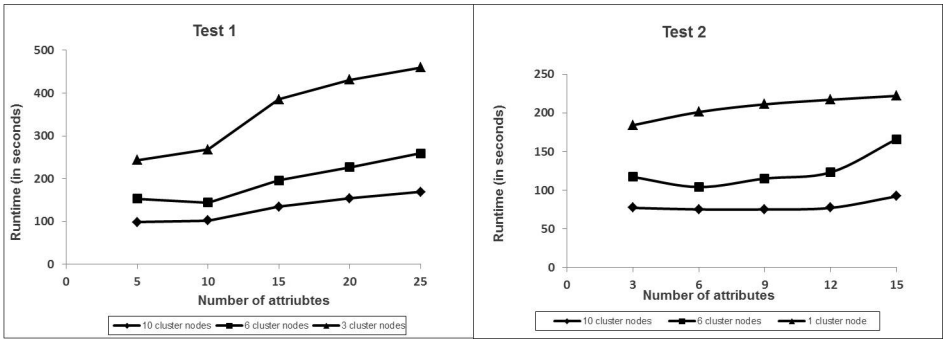

Fig. 5. Size up behaviour of Parallel Random Prism with respect to the number of features. Headings Test 1 and Test 2 refer to the test datasets in Table 1. These datasets have in this case been appended to themselves in order to increase the number of attributes, while keeping the concept stable.

Note that for this set of size-up experiments there is no setup with only one cluster node. The reason for this is that we used the original number of data features for both datasets, which simply exceeds the computational capabilities of one cluster node after the bagging procedure for 100 base classifiers.

In general we can observe a nice size-up that is close to being linear with respect to the number of features. These results clearly support the theoretical average linear behaviour.

The speed-up factors recorded for Parallel Random Prism, on both test datasets and for different numbers of cluster nodes (up to the 10 available) are displayed in Figure 6. The theoretical ideal speed-up factors are plotted as a dashed line. It can be seen that the speed-up factors achieved are very close to the ideal linear case. This almost ideal speed-up has been veryfied by linear regression equations also depicted in Figure 6. There is a small discrepancy between the ideal case and the actual speed-up factors, the more cluster nodes are used. However, this discrepancy is expected and can be explained by the non parallel part of Parallel Random Prism as mentioned in the previous section, which is the term $\sum_{i=1}^{b} T_{asm,i}$ and the communication overhead, which is $T_{comdat} \cdot p$ in the equation for the speed-up of Parallel Random Prism. It is expected that there will be an upper limit of the number of cluster nodes that are beneficial to reducing the runtime. However, considering the low discrepancy after using 10 cluster nodes suggests that the impact of $\sum_{i=1}^{b} T_{asm,i}$ and $T_{comdat} \cdot p$ is not very high and thus the experiments are far from using the maximum number of cluster nodes that are still beneficial to lowering the runtime. This is consistent with the theoretical speed-up analysis in the previous section. Please note that the theoretical and empirical analysis presented in this paper

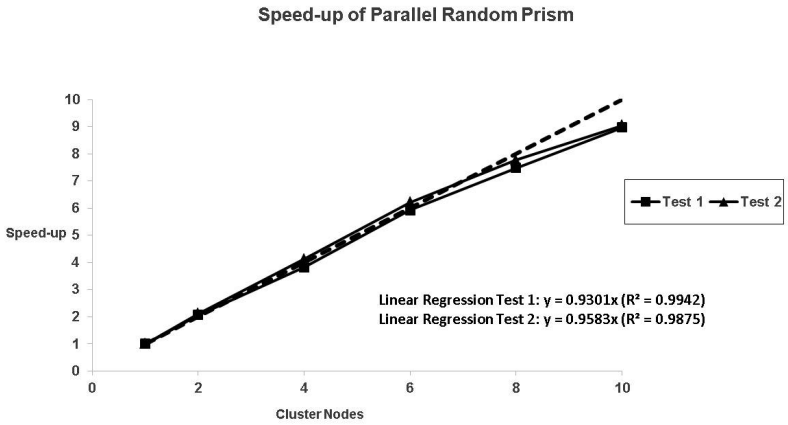

Fig. 6. The Speed-up factors for of Parallel Random Prism. The dashed line represents the theoretical ideal speedup. Linear regression equations and R^2 are displayed for the two test cases.

focuses on the algorithm rather than the version of MapReduce being used. If the sample constructed constructed for the R-PrismTCS classifier is bigger than the HDFS block size additional communication overhead will be incurred and the less speedup can be achieved. The samples constructed in the experiments outlined in this paper were not bigger than the HDFS block size.

Loosely speaking, Parallel Random Prism indeed exhibits an linear scalability with respect to the number of training instances and the number of features. Furthermore, the algorithm also shows a near linear speed-up factor.

The current implementation of Parallel Random Prism is bound in its maximum parallelism by the number of R-PrismTCS classifiers utilised. However, R-PrismTCS classifiers could also be parallelised. The Parallel Modular Classification Rule Induction (PMCRI) framework [19] for parallelising, amongst others, the PrismTCS [5] classifier, can be used for parallelising the R-PrismTCS classifier also. This is due to the similarity of the R-PrismTCS and PrismTCS classifiers. However, this is outside the scope of this paper.

6 Conclusions

This paper presented work on a novel, well-scaling ensemble classifier called Parallel Random Prism. Ensemble classifiers exhibit a very high predictive accuracy compared with standalone classifiers, especially in noisy domains. However, this increase in performance is at the expense of computational efficiency due to data replication and the induction of multiple classifiers. Thus ensemble classifiers applied on modest size training data already challenge the computational hardware. Section 2 highlighted alternative base classifiers to decision trees (on

which most ensemble classifiers are based), in particular the Prism approach. The PrismTCS standalone classifier often outperforms decision trees when applied to noisy data, and hence is a good candidate base classifier for ensemble classifiers. Section 2 proposed the Random Prism ensemble learner with the PrismTCS based R-PrismTCS base classifier. It summarised results concerning classification accuracy and gave an initial empirical estimate of Random Prism's runtime requirements. Section 3 also highlighted a parallel version of Random Prism using the Hadoop implementation of the MapReduce programming paradigm. Essentially multiple R-PrismTCS base classifiers are executed concurrently on p computing nodes in a Hadoop cluster. The only aspects of Random Prism that are not parallelised are the inexpensive combining procedure of the individual classifiers and the distribution of the original training data over the cluster. Section 4 gave a theoretical complexity analysis of Random Prism and a theoretical scalability analysis of Parallel Random Prism. The parallel version of Random Prism was examined in terms of its runtime with respect to the number of computing nodes used. In general a close to linear scalability was expected, as the main part of the workload, the base classifier induction was parallelised. However, the data communication to the cluster nodes at the beginning and the combining procedures were not parallelised, hence an upper limit of beneficial computing nodes was expected. Section 5 further supported the theoretical analysis with empirical results. In these results Parallel Random Prism's linear scalability with respect to the number of training instances and features was confirmed. These results also showed that Parallel Random Prism exhibits an almost ideal speed-up for up to 10 cluster nodes with a slightly increasing deterioration the more cluster nodes are utilised. The results suggested that there is an upper limit (due to the non-parallel parts of Parallel Random Prism). However, the results also suggested that the cluster is far away from its maximum number of beneficial cluster nodes.

References

1. Hadoop, http://hadoop.apache.org/ 2014.
2. Jaume Bacardit and Natalio Krasnogor. The infobiotics PSP benchmarks repository. Technical report, 2008.
3. K. Bache and M. Lichman. UCI machine learning repository, 2013.
4. 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.
5. 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.
6. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
7. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
8. J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.
9. 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.

10. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.
11. Yves Grandvalet. Bagging equalizes influence. *Machine Learning*, 55(3):251–270, 2004.
12. J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier, 2011.
13. John L Hennessy and David A Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, USA, third edition, 2003.
14. Tin Kam Ho. Random decision forests. *Document Analysis and Recognition, International Conference on*, 1:278, 1995.
15. Kai Hwang and Fay A Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Book Co., international edition, 1987.
16. Ting Liu, Charles Rosenberg, and Henry A. Rowley. Clustering billions of images with large scale nearest neighbor search. In *Proceedings of the Eighth IEEE Workshop on Applications of Computer Vision*, WACV '07, page 28, Washington, DC, USA, 2007. IEEE Computer Society.
17. 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.
18. Ross J Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
19. Frederic Stahl and Max Bramer. Computationally efficient induction of classification rules with the pmcri and j-pmcri frameworks. *Knowledge-Based Systems*, 2012.
20. Frederic Stahl and Max Bramer. Random prism: a noise-tolerant alternative to random forests. *Expert Systems*, 2013.
21. Frederic Stahl, Max Bramer, and Mo Adda. Parallel rule induction with information theoretic pre-pruning. In Max Bramer, Richard Ellis, and Miltos Petridis, editors, *Research and Development in Intelligent Systems XXVI*, pages 151–164. Springer London, 2010.
22. Frederic Stahl, David May, and Max Bramer. Parallel random prism: A computationally efficient ensemble learner for classification. In Max Bramer and Miltos Petridis, editors, *Research and Development in Intelligent Systems XXIX*, pages 21–34. Springer London, 2012.
23. Raja Tlili and Yahya Slimani. A hierarchical dynamic load balancing strategy for distributed data mining. *International Journal of Advanced Science and Technology*, 2012.
24. Ian.H. Witten, Eibe Frank, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.