Random Prism: An Alternative to Random Forests

Frederic Stahl and Max Bramer

Abstract Ensemble learning techniques generate multiple classifiers, so called base classifiers, whose combined classification results are used in order to increase the overall classification accuracy. In most ensemble classifiers the base classifiers are based on the Top Down Induction of Decision Trees (TDIDT) approach. However, an alternative approach for the induction of rule based classifiers is the Prism family of algorithms. Prism algorithms produce modular classification rules that do not necessarily fit into a decision tree structure. Prism classification rulesets achieve a comparable and sometimes higher classification accuracy compared with decision tree classifiers, if the data is noisy and large. Yet Prism still suffers from overfitting on noisy and large datasets. In practice ensemble techniques tend to reduce the overfitting, however there exists no ensemble learner for modular classification rule inducers such as the Prism family of algorithms. This article describes the first development of an ensemble learner based on the Prism family of algorithms in order to enhance Prism's classification accuracy by reducing overfitting.

1 Introduction

One of the most well-known ensemble learning methods is the Random Forests (RF) classifier from Breiman [7]. RF is inspired by the Random Decision Forests (RDF) approach from Ho [15]. Ho argues that traditional trees often cannot be grown over a certain level of complexity without risking a loss of generalisation caused by overfitting on the training data. Ho proposes inducing multiple trees in randomly selected

Frederic Stahl
School of Computing, Buckingham Building, Lion Terrace, PO1 3HE e-mail: Frederic.Stahl@port.ac.uk

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

subsets of the feature space. He claims that the combined classification will improve, as the individual trees will generalise better on the classification for their subset of the feature space. Ho evaluates his claims empirically. RF makes use of the basic RDF approach by combining it with Breiman's bagging (**B**ootstrap **agg**regat**ing**) method [6]. Bagging is intended to improve a classifier's stability and classification accuracy. A classifier is unstable if a small change in the training set causes major variations in the classification.

Research on ensemble learning technologies for classification in order to overcome overfitting is still ongoing. For example [13] generate ensembles of heterogeneous classifiers using stacking. [11] proposed a framework for generating hundreds of thousands of classifiers in parallel in a distributed environment using small subsamples of the dataset. Chan and Stolfo's [9, 10] Meta-Learning framework partitions the data into subsamples that fit into the memory of a single machine and developed a classifier in each subset separately. These classifiers are then combined using various algorithms in order to create a final classifier. This can easily be run in parallel using the independent multi-sample mining approach [19]. A recently developed prototype of ensemble learners, based on Hoeffding Trees [17] and incremental Naive Bayes, for the classification of datastreams in an ad hoc network of mobile phones is discussed here [27, 26]. The overall data mining framework is called 'Pocket Data Mining' [30]. Pocket Data Mining employs weighted majority voting in order to combine the different classifiers induced on different mobile phones. This work uses the terms 'ensemble learner' and 'ensemble classifier' interchangeably, referring to ensemble learners for classification unless stated otherwise.

There are two general approaches to the induction of classification rules, the 'divide and conquer' approach, also known as TDIDT [20, 21] and the 'separate and conquer' approach [31]. 'Divide and conquer' induces classification rules in the intermediate representation of a decision tree. 'Separate and conquer' induces a set of *IF..THEN* rules rather than a decision tree. 'Separate and conquer' can be traced back to Michalski's AQ system in the 1960s [16]. However the most notable development using the 'separate and conquer' approach is the Prism family of algorithms [8, 3, 4]. It produces modular rules that do not necessarily fit into a decision tree. It produces a comparable classification accuracy to and in some cases outperforms decision trees, especially in noisy domains. Recent developments on the Prism family of algorithms includes frameworks for parallelising Prism algorithms for rule induction on massive datasets [23, 25, 24] and rule pruning methods in order to reduce overfitting [28, 4]. In general Prism algorithms have been shown to be less vulnerable to overfitting compared with decision tree classifiers, especially if there is noise in the data and missing values [3]. Yet most ensemble learning approaches are either based on decision trees or a heterogeneous setup of base classifiers. Some ensemble approaches consider heterogeneous classifiers, such as Meta-Learning [9, 10], yet in practice their application mostly makes use of algorithms that follow the 'divide and conquer' approach.

The fact that Prism classifiers tend to overfit less compared with decision trees motivates the development of ensemble learners based on Prism algorithms. This paper presents the first attempt to build a Prism based ensemble learner inspired

from RF called *Random Prism* in order to improve Prism's classification accuracy further. A prototype implementation is evaluated empirically. This paper is structured as follows: Section 2 introduces the Prism family of algorithms in comparison with decision tree classifiers and describes the newly developed *Random Prism* ensemble learner; Section 3 evaluates Random Prism on several datasets and compares it with a standalone Prism classifier. Section 4 highlights some ongoing and future work, notably some variations of the Random Prism approach and a parallel version of Random Prism. Section 5 concludes the paper with a brief summary and some concluding remarks.

2 Random Prism

This section describes our Random Prism ensemble learner. It first introduces the Prism Family of algorithms in Section 2.1 and compares them with the 'divide and conquer' approach used by RF. Section 2.2 then highlights the Random Prism approach based on the RF ensemble learner.

2.1 The Prism Family of Algorithms

As mentioned in Section 1, the representation of classification rules differs between the 'divide and conquer' and 'separate and conquer' approaches. Rule sets generated by the 'divide and conquer' approach are in the form of decision trees whereas rules generated by the 'separate and conquer' approach are modular. Modular rules do not necessarily fit into a decision tree and normally do not. The rule representation of decision trees is the main drawback of the 'divide and conquer' approach, for example rules such as:

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

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

cannot be represented in a tree structure as they have no attribute in common. Forcing these rules into a tree will require the introduction of additional rule terms that are logically redundant, and thus result in unnecessarily large and confusing trees [8]. This is also known as the replicated subtree problem [31]. Cendrowska illustrates the replicated subtree using the two example rules above in [8]. Cendrowska assumes that the attributes in the two rules above comprise three possible values and both rules predict class *x*, all remaining classes being labelled *y*. The simplest tree that can express the two rules is shown in Figure 1. The total set of rules that predict class *x* encoded in the tree is:

IF A = 1 AND B = 1 THEN Class = x
IF A = 1 AND B = 2 AND C = 1 AND D = 1 THEN Class = x

IF A = 1 AND B = 3 AND C = 1 AND D = 1 THEN Class = x
IF A = 2 AND C = 1 AND D = 1 THEN Class = x
IF A = 3 AND C = 1 AND D = 1 THEN Class = x

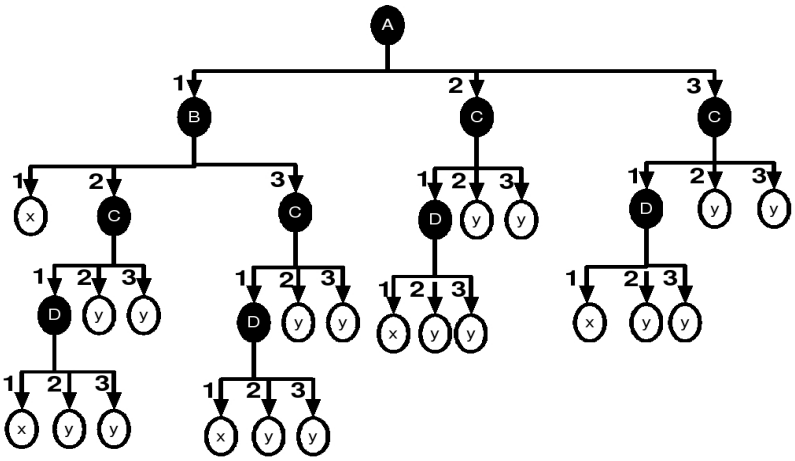

Fig. 1 Cendrowska's replicated subtree example.

Cendrowska argues that situations such as this which cause trees to be needlessly complex make the tree representation unsuitable for expert systems and may require unnecessary expensive tests by the user [8].

'Separate and conquer' algorithms avoid the replicated subtree problem by inducing directly sets of 'modular' rules, avoiding unnecessarily redundant rule terms that are induced just for the representation in a tree structure. The basic 'separate and conquer' approach can be described as follows, where the statement

```
Rule_set rules = new Rule_set();
```

creates a new rule set:

```
Rule_Set rules = new Rule_set();
While Stopping Criterion not satisfied{
    Rule = Learn_Rule;
    Remove all data instances covered from Rule;
    rules.add(rule);
}
```

The *Learn_Rule* procedure generates the best rule for the current subset of the training data where best is defined by a particular heuristic that may vary from algorithm to algorithm. The stopping criterion is also dependent on the algorithm used. After inducing a rule, the rule is added to the rule set, all instances that are covered by the rule are deleted and a new rule is induced on the remaining training instances.

In Prism each rule is generated for a particular Target Class (TC). The heuristic Prism uses in order to specialise a rule is the probability with which the rule covers

the TC in the current subset of the training data. The stopping criterion is fulfilled as soon as there are no training instances left that are associated with the TC.

Cendrowska's original Prism algorithm selects one class as the TC at the beginning and induces all rules for that class. It then selects the next class as TC and resets the whole training data to its original size and induces all rules for the next TC. This is repeated until all classes have been selected as TC. Variations exist such as PrismTC [5] and PrismTCS (Target Class Smallest first) [4]. Both select the TC anew after each rule induced. PrismTC always uses the majority class and PrismTCS uses the minority class. Both variations introduce an order in which the rules are induced, where there is none in the basic Prism approach. However unpublished experiments by the current authors show that the predictive accuracy of PrismTC cannot compete with that of Prism and PrismTCS. PrismTCS does not reset the dataset to its original size and thus is faster than Prism. It produces a high classification accuracy and also sets an order in which the rules should be applied to the test set.

The basic PrismTCS algorithm is outlined below where A_x is a possible attribute value pair and D is the training dataset. The statement

```
Rule_set rules = new Rule_set();
```

creates a new rule set which is a list of rules and the line

```
Rule rule = new Rule(i);
```

creates a new rule with class i as classification. The statement

```
rule.addTerm(Ax);
```

will append a rule term to the rule and

```
rules.add(rule);
```

adds the finished rule to the rule set.

```
        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;
        }
```

We will concentrate here on the more popular PrismTCS approach but all techniques and methods outlined here can be applied to any member of the Prism family.

2.2 Random Prism Classifier

The Random Prism classifier is based on the principles of the RF ensemble learner, hence this section first introduces the Random Forests classifier briefly and then discusses the new Random Prism ensemble classifier.

As mentioned in Section 1 RF are inspired by the RDF approach from Ho [15]. RDF induces multiple trees in randomly selected subsets of the feature space in order to make the trees generalise better. RF uses RDF's approach plus bagging [6] in order to improve the classifiers' accuracy and stability. Bagging means that a sample with replacement is taken for the induction of each tree.

The basic principle of RF is that it grows a large number of decision trees (a forest) on samples produced by bagging, using a random subset of the feature space for the evaluation of splits at each node in each tree. If there is a new data instance to be classified, then each tree is used to produce a prediction for the new data instance. RF then labels the new data instance with the class that achieved the majority of the 'votes'.

The Random Prism ensemble learner's ingredients are the RDF's random feature subset selection, RF's bagging and PrismTCS as base classifier.

Using Prism as base classifier is motivated by the fact that Prism is less vulnerable to clashes, missing values and noise in the dataset and in general tends to overfit less compared with decision trees [3] which are used in RF and RDF. In particular PrismTCS is used, as PrismTCS is also computationally more efficient than the original Prism while in some cases producing a better accuracy [29]. A good computational efficiency is needed as ensemble classifiers induce multiple classifiers and thus place a high computational demand on CPU time. In the context of Random Prism, the terms PrismTCS and Prism may be used interchangeably in this paper, both referring to PrismTCS unless stated otherwise.

Given a training dataset D, using bagging a sample D_i if i is the *ith* classifier is created, using random sampling with replacement [6]. This means that the data samples may overlap, as in D_i a data instance may occur more than once or may not be included at all. From each D_i a classifier C_i is induced. In order to classify a new data instance, each C_i predicts the class, and the bagged classifier counts the votes and labels the data instance with the class that achieved the majority of the votes. An ensemble classifier created using bagging often achieves a higher accuracy compared with a single classifier induced on the whole training dataset D and if it achieves a worse accuracy it is often still close to the single classifier's accuracy [14]. The reason for the increased accuracy of bagged classifiers lies in the fact that the composite classifier model reduces the variance of the individual classifiers [14]. The most commonly used bootstrap model for bagging is to take a sample of size n if n is the number of instances. This will result in samples that contain on average 63.2% of the original data instances. The fact that bagged classifiers can achieve a higher accuracy than a single classifier induced on the whole dataset D, as mentioned above, motivates the use of bagging in the Random Prism ensemble classifier proposed here.

The RDF approach by Ho [15] induces multiple trees on randomly selected subsets of the feature space. Again a composite model is generated and it has been shown in [15] that they generalise better than a single tree induced on the complete feature space, as the are less prone to overfitting. Inspired from RDF, Breiman's RF randomly selects a subset of the feature space for each node of each tree. Feature subset selection similar to the one used in RF is incorporated in Random Prism as well, inspired from the fact that random feature subset selection generalises the ensemble classifier better and thus makes it likely to produce a higher classification accuracy.

The pseudo code below describes the adapted version of PrismTCS for the use in Random Prism based on PrismTCS's pseudo code in Section 2.1. *M* is 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);
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;
        }
```

The pseudo code above is essentially PrismTCS incorporating RDF's and RF's random feature subset selection. For the induction of each rule term for each rule, a fresh random subset of the feature space is drawn. Also the number of features considered for each rule term is a random number between 1 and *M*. The PrismTCS version above is called *R-PrismTCS*, *R* for denoting **R**andom sample and feature selection.

The basic Random Prism approach is outlined in the pseudo code below, where *k* is the number of *R-PrismTCS* classifiers to be induced and *i* is the *ith* classifier:

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

Please note that in the Random Prism pseudo code above not only a set of classifiers is created but also a set of weights. Random Prism does not employ a simple voting system like RF or RDF, but a weighted majority voting system as in the

Pocket Data Mining System [27, 26], where each vote is weighted according to the corresponding classifier's accuracy on the test data. As mentioned earlier in this section, the sampling method used for each classifier selects approximately 63.2% percent of the total number of data instances, which leaves approximately 36.8% of the total number of data instances which are used to calculate the individual *R-PrismTCS* classifier's accuracy and thus weight. Also the user of the Random Prism classifier can define a threshold *N*, which is the precentage of classifiers to be used for prediction. Random Prism will always select those classifiers with the highest weights.

For example consider classifiers and weights listed in Table 1.

Table 1 Example data for weighted majority voting

Classifier	Weight
A	0.55
B	0.65
C	0.55
D	0.95
E	0.85

Assume that the classifiers in Table 1 are already the best classifiers selected according to the user's defined threshold. Further assume that for a new unseen data instance classifiers *A, B* and *C* predict class *Y* and classifiers *D* and *E* predict class *X*. Random Prism's weighted majority vote for class *Y* is 1.75 (i.e. $0.55+0.65+0.55$) and for class *X* is 1.80 (i.e. $0.95+0.85$). Thus Random Prism will label the data instance with class *X*.

The *R-PrismTCS* pseudo code above does not take pruning into consideration, however a pre-pruning method *J-pruning* presented in [4] is implemented in *R-PrismTCS* in order further generalise the base classifiers. J-pruning is based on the J-measure. According to Smyth and Goodman [22] the average information content of a rule of the form *IF Y = y THEN X = x* can be quantified by the following equation:

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

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

$$j(X;Y=y) = p(x \mid y) \cdot log_2(\tfrac{p(x|y)}{p(x)}) + (1-p(x \mid y)) \cdot log_2(\tfrac{(1-p(x|y))}{(1-p(x))}) \quad (2)$$

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

3 Evaluation of Random Prism Classification

Random Prism has been evaluated on 15 different datasets retrieved from the UCI data repository [2]. For each dataset a test and a training set has been created using random sampling without replacement. The training set comprises 70% of the total data instances. Please note that the training set is sampled again by each R-PrismTCS base classifier, in order to incorporate bagging. Hence, as stated in Section 2.2 approximately 63.2% of the training data is used for the actual training and 36.8% is used to calculate the individual classifiers' weights. The percentage of the best classifiers to be used was 10% and the total number of R-PrismTCS classifiers induced was 100 for each dataset.

Table 2 shows the accuracy achieved using Random Prism classifier and the accuracy achieved using a single PrismTCS classifier.

Table 2 Accuracy of Random Prism compared with PrismTCS.

Dataset	Accuracy PrismTCS	Accuracy Random Prism
monk1	0.79	1.0
monk3	0.98	0.99
vote	0.94	0.95
genetics	0.70	0.88
contact lenses	0.95	0.91
breast cancer	0.95	0.95
soybean	0.88	0.65
australian credit	0.89	0.92
diabetes	0.75	0.89
crx	0.83	0.86
segmentation	0.79	0.71
ecoli	0.78	0.78
balance scale	0.72	0.86
car evaluation	0.76	0.71
contraceptive method choice	0.44	0.54

As can be seen in Table 2 Random Prism outperforms PrismTCS in 9 out of 15 cases; in two cases Random Prism achieved the same accuracy as PrismTCS; and in only 4 cases Random Prism's accuracy was below that of PrismTCS. However, looking into these four cases with a lower accuracy, which is for datasets 'car evaluation', 'segmentation', 'soybean' and 'contact lenses', it can be seen that the accuracies for 'car evaluation' and 'contact lenses' is still very close the PrismTCS's accuracy. In

general Random Prism outperforms its single classifier version PrismTCS in most cases and in the remaining cases its accuracy is often very close to PrismTCS's accuracy.

4 Ongoing and Future Work

Ongoing and future work comprises a distributed / parallel version of Random Prism and several variations of the Random Prism approach itself.

4.1 Parallel Random Prism Classifier

Random Prism like any other ensemble learner has a higher demand on CPU time than its single classifier version. Table 3 lists the runtimes of PrismTCS and Random Prism for the evaluation experiments outlined in Section 3. As ensemble learners are designed to reduce overfitting, they should be able to be executed on larger datasets as well, as the likelihood that noisy data instances are present is higher the larger the training data is.

Table 3 Runtime of Random Prism on 100 base classifiers compared with a single PrismTCS classifier in milli seconds.

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

It can be seen that as expected the runtimes are much larger for Random Prism than for PrismTCS. This is because Random Prism induces 100 base classifiers whereas PrismTCS is only a single classifier. One would expect the runtimes of Random Prism to be 100 times longer than for PrismTCS as Random Prism induces 100 base classifiers, however the runtimes are much shorter than expected. The rea-

son for this is that the base classifiers use a subset of the feature space and thus have fewer features to scan for the induction of each rule term.

Future work will address the problem of scalability of the Random Prism classifier. Google's Parallel Learner for Assembling Numerous Ensemble Trees (PLANET) system [18] addresses this problem in the context of decision tree based ensemble classifiers using the MapReduce [12] model of distributed computation. MapReduce builds a cluster of computers for a two-phase distributed computation on large volumes of data. First in the map-phase the dataset is split into disjoint subsets, which are assigned together with a user specified map function to workers (mappers) in the MapReduce cluster. Each mapper then applies the map function on its data. The output of the map function (a key-value pair) is then grouped and combined by a second kind of worker, the reducers, using a user defined reduce function.

For Random Prism the MapReduce model will be used to distribute the induction of the R-PrismTCS base classifiers using mappers. The individual R-PrismTCS classifiers are then combined using the reducers to the final Random Prism Classifier. Thus the CUP intense part, the induction of many base classifiers can easily be distributed to a computing cluster of workstations. A open source implementation of the MapReduce model called Hadoop is available [1].

4.2 Variations of the Random Prism Ensemble Classifier

There are many possible variations of the Random Prism approach that may achieve better classification accuracy, for example different versions of Prism could be used as base classifiers. Also it would be possible to use a diverse mix of all existing Prism classifiers, such as Prism, PrismTC or PrismTCS. Some Prism classifiers may perform well on certain samples, some may perform worse, thus a larger variety of Prism classifiers per sample may well increase Random Prism's classification accuracy.. Also it is possible to use several Prism and decision tree base classifiers for each sample.

4.3 Intelligent Voting System

Random Prism's classification accuracy may be further improved by employing a more intelligent voting system. For example a classifier may have in general a moderate predictive accuracy. However, concerning its predictions for class *A*, the classifier may have a very high predictive accuracy. Such cases could be addressed by calculating individual weights for each class for this particular classifier. Implementing more than one weight for a classifier must also be addressed in the selection of the best classifiers according to a user defined percentage. A similar approach called 'Combining' has been used by the Meta-Learning system [9, 10].

5 Conclusions

This work presents the Random Prism ensemble classifier based on the Prism family of algorithms as base classifier. Most ensemble learners are based on decision trees as base classifiers and aim to reduce the overfitting of the model in order to achieve a higher classification accuracy. However alternative base classifiers exist, such as the Prism family of algorithms. It has been discussed that Prism algorithms already perform better on noisy datasets compared with decision trees, as they tend to overfit less. The motivation behind Random Prism is that an ensemble classifier based on the Prism family of algorithms may further reduce the overfitting and thus achieve a higher classification accuracy compared with single Prism classifiers.

First the Prism family of algorithms has been introduced and compared with decision trees and next the well known Random Forests approach has been reviewed. Random Prism is inspired from the Prism family of algorithms, the Random Decision Forests and Random Forests approaches. Random Prism uses the PrismTCS classifier as base classifier with some modifications called R-PrismTCS. The modifications were in order to use the Random Decision Forests' feature subset selection approach. Random Prism also incorporates J-pruning for R-PrismTCS and Random Forests' bagging approach. Contrary to Random Forests and Random Decision Forests, Random Prism uses a weighted majority voting system instead of a plain majority voting system, in order to take the individual classifier's classification accuracy into account. Also Random Prism does not take all classifiers into account, the user can define the percentage of classifiers to be used for classification. Random Prism will select only the classifiers with the highest classification accuracy for the classification task.

Random Prism has been evaluated on 15 datasets from the UCI repository and has been shown to produce a better classification accuracy on 9 cases compared with PrismTCS. In two cases the classification accuracy was the same as for PrismTCS. In two further cases the classification accuracy was slightly below PrismTCS's accuracy and only in two cases was it much worse than PrismTCS's accuracy.

Ongoing work on Random Prism comprises the development of a distributed / parallel version in order to make Random Prism scale better on large datasets. For this the MapReduce framework is considered in order to distribute the induction of the individual classifiers to different machines in a cluster of workstations. This could be realised using a open source implementation of MapReduce called Hadoop. Furthermore a variety of Random Prism versions are planned, comprising different Prism classifiers as base classifiers or even hybrid ensemble learners comprising different versions of Prism in one ensemble learner or possibly a mix of decision tree and Prism classifiers.

Acknowledgements We would like to acknowledge Mohamed Medhat Gaber for his advice during the implementation of the Random Prism algorithm.

References

1. Hadoop, http://hadoop.apache.org/mapreduce/ 2011.
2. 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.
3. 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.
4. 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.
5. M A Bramer. Inducer: a public domain workbench for data mining. *International Journal of Systems Science*, 36(14):909–919, 2005.
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. Experiments on multistrategy learning by meta learning. In *Proc. Second Intl. Conference on Information and Knowledge Management*, pages 314–323, 1993.
10. 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.
11. Nitesh V. Chawla, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *J. Mach. Learn. Res.*, 5:421–451, December 2004.
12. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.
13. Saso Dzeroski and Bernard Zenko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54:255–273, 2004.
14. Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.
15. Tin Kam Ho. Random decision forests. *Document Analysis and Recognition, International Conference on*, 1:278, 1995.
16. R S Michalski. On the Quasi-Minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing*, pages 125–128, Bled, Yugoslavia, 1969.
17. Domingos P. and Hulten G. Mining high-speed data streams. In *In International Conference on Knowledge Discovery and Data Mining*, pages 71–81, 2000.
18. 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.
19. Foster Provost. Distributed data mining: Scaling up and beyond. In *Advances in Distributed and Parallel Knowledge Discovery*, pages 3–27. MIT Press, 2000.
20. R J Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
21. Ross J Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
22. P. Smyth and R M Goodman. An information theoretic approach to rule induction from databases. *Transactions on Knowledge and Data Engineering*, 4(4):301–316, 1992.
23. F T Stahl, M A Bramer, and M Adda. PMCRI: A parallel modular classification rule induction framework. In *MLDM*, pages 148–162. Springer, 2009.
24. Frederic Stahl, Max Bramer, and Mo Adda. J-PMCRI: a methodology for inducing pre-pruned modular classification rules. *IFIP Advances in Information and Communication Technology*, 331:47–56, 2010.

25. Frederic Stahl, Max Bramer, and Mo Adda. Parallel rule induction with information theoretic pre-pruning. In *Research and Development in Intelligent Systems XXVI*, volume 4, pages 151–164. Springerlink, 2010.
26. Frederic Stahl, Mohamed Medhat Gaber, Max Bramer, and Phillip S. Yu. Distributed hoeffding trees for pocket data mining. In *The 2011 International Conference on High Performance Computing and Simulation*, Istanbul, Turkey, in Press (2011).
27. Frederic Stahl, Mohamed Medhat Gaber, Han Liu, Max Bramer, and Phillip S. Yu. Distributed classification for pocket data mining. In *19th International Symposium on Methodologies for Intelligent Systems*, Warsaw, Poland, in Press (2011). Springer.
28. Frederic T. Stahl and Max Bramer. Induction of modular classification rules: Using jmax-pruning. In *SGAI Conf.'10*, pages 79–92, 2010.
29. Frederic T. Stahl, Max Bramer, and Mo Adda. Parallel induction of modular classification rules. In *SGAI Conf.*, pages lookup–lookup. Springer, 2008.
30. Frederic T. Stahl, Mohamed Medhat Gaber, Max Bramer, and Philip S. Yu. Pocket data mining: Towards collaborative data mining in mobile computing environments. In *ICTAI (2)'10*, pages 323–330, 2010.
31. I H Witten and F Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.