University of Reading

Random Prism: a noise-tolerant alternative to Random Forests

Article

Accepted Version

Stahl, F. ORCID: https://orcid.org/0000-0002-4860-0203 and Bramer, M. (2014) Random Prism: a noise-tolerant alternative to Random Forests. Expert Systems, 31 (5). pp. 411-420. ISSN 1468-0394 doi: 10.1111/exsy.12032 (special issue on innovative techniques and applications of artificial intelligence) Available at https://centaur.reading.ac.uk/32914/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

To link to this article DOI: http://dx.doi.org/10.1111/exsy.12032

Publisher: Wiley-Blackwell

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

CentAUR

University of Reading

Central Archive at the University of Reading

Reading's research outputs online

Random Prism: A Noise Tolerant Alternative to Random Forests

Frederic Stahl and Max Bramer

Abstract Ensemble learning can be used to increase the overall classification accuracy of a classifier by generating multiple base classifiers and combining their classification results. A frequently used family of base classifiers for ensemble learning are decision trees. However, alternative approaches can potentially be used, such as the Prism family of algorithms which also induces classification rules. Compared with decision trees Prism algorithms generate modular classification rules that cannot necessarily be represented in the form of a decision tree. Prism algorithms produce a similar classification accuracy compared with decision trees. However, in some cases, for example if there is noise in the training and test data, Prism algorithms can outperform decision trees by achieving a higher classification accuracy. However, Prism still tends to overfit on noisy data; hence ensemble learners have been adopted in this work to reduce the overfitting. This article describes the development of an ensemble learner using a member of the Prism family as the base classifier in order to reduce the overfitting of Prism algorithms on noisy datasets. The developed ensemble classifier is compared with a stand-alone Prism classifier in terms of classification accuracy and resistance to noise.

1 Introduction

Ho's Random Decision Forests (RDF) [15] and Breiman's Random Forests (RF) [7], which was inspired by RDF, are two of the best known ensemble

Frederic Stahl
University of Reading, School of Systems Engineering, PO Box 225 Whiteknights, Reading, RG6 6AY e-mail: F.T.Stahl@reading.ac.uk

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

learning methods. Ho claims that a decision tree cannot grow beyond a certain level without risking a loss of generalisation due to overfitting and proposes inducing multiple trees on randomly selected features. According to his theory, a combined classification will produce a better classifier as the individual trees generalise better on their subset of the feature space. Ho evaluates his theory empirically. Breiman's RF is based on Ho's RDF approach but also incorporates bagging (**B**ootstrap **agg**regat**ing**) [6], which is used to improve the classifier in terms of its stability and classification accuracy. An unstable classifier shows considerable variations to small changes in the training set. However, alternative partitioning strategies to bagging have been developed, such as [33, 31].

Like most rule based classifiers and ensemble classifiers, RDF and RF are based on the 'divide and conquer' rule induction approach. 'Divide and conquer' induces classification rules in the intermediate form of a decision tree [21, 22]. A more general approach to inducing classification rules is the 'separate and conquer' [32] approach, which induces a set of *IF..THEN* rules that do not necessarily fit into a decision tree representation. This alternative approach to decision trees can be traced back to the AQ system in the 1960s [17]. However, modern representatives exist such as the Prism family of algorithms [25].

This paper presents and evaluates an ensemble classifier, based on the 'separate and conquer' approach, called *Random Prism* [24]. Random Prism is inspired by RF, in order to improve Prism's classification accuracy further. The prototype is evaluated empirically regarding its classification accuracy in general and in noisy domains.

This paper is structured as follows: Section 2 highlights related work in the context Random Prism. Section 3 introduces the Prism family of algorithms in comparison with decision tree classifiers and describes the newly developed *Random Prism* ensemble learner; Section 4 evaluates Random Prism on several datasets and compares it with a standalone Prism classifier in terms of classification accuracy and tolerance to noise. Section 5 highlights some ongoing and future work, notably some variations of the Random Prism approach and a parallel version of the Random Prism ensemble classifier. Section 6 closes the paper with a brief summary and some concluding remarks.

2 Related Work

Besides the RF and RDF approaches mentioned in Section 1, ongoing research on ensemble learners, undertaken with the aim of overcoming overfitting, comprises Dzeroski's work on generating ensembles of heterogeneous classifiers using stacking [13]. The authors of [11] propose a parallel approach to generating hundreds of thousands of classifiers on small samples of the dataset in a distributed computing environment. Chan and Stolfo propose

the Meta-Learning framework [9, 10] which combines classifiers using various algorithms in order to create a final classifier, which could easily be parallelised using the multi-sample mining approach [20]. A recent development in ensemble learning is 'Pocket Data Mining' for mining data streams using heterogeneous base classifiers executed on different devices in an ad hoc network of smart phones [28]. Pocket Data Mining's combining method is based on weighted majority voting. Also regarding data stream mining, the authors of [16] and [18] have developed ensemble techniques aiming to detect concept drift. Ensemble learning methods are not necessarily classifiers. However, in the literature the terms ensemble learner and ensemble classifier are often used interchangeably referring to an ensemble classifier, and this paper is no exception.

As mentioned in Section 1 most ensemble learning methods are based on 'divide and conquer' algorithms for generating the base classifiers. There are heterogeneous ensemble approaches such as Meta-Learning [9, 10] that use different base classifiers. However, in practice these heterogeneous base classifiers are different versions of decision trees and hence use the 'divide and conquer' approach. A notable family of algorithms that follows the alternative 'separate and conquer' approach is the Prism family of algorithms [8, 3, 4]. A recent development of the Prism family is the parallelisation of Prism including a parallelised pre-pruning facility [27], an improved pre-pruning method, Jmax-pruning [26] and an adaptive version of Prism for data stream classification called 'eRules' [30]. Generally members of the Prism family produce a comparable classification accuracy to decision trees. However, in some cases they outperform decision trees, especially when the training and test datasets are noisy or there are missing values [3]. The fact that Prism classifiers tend to feature less overfitting in comparison with decision trees is the motivation for developing the Random Prism ensemble classifier aiming to improve Prism's predictive accuracy further.

3 Random Prism

This section describes our Random Prism ensemble learner. It first introduces the Prism Family of algorithms in Section 3.1 and compares them with the 'divide and conquer' approach used by RF. Section 3.2 then highlights the Random Prism approach based on the RF ensemble learner.

3.1 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 attributes 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 [32]. 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

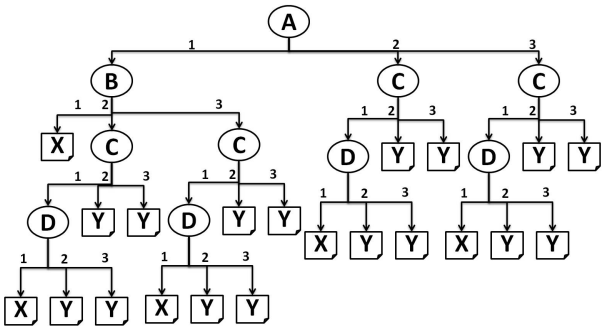

Fig. 1 Cendrowska's replicated subtree example.

Cendrowska argues that situations such as this can 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 directly inducing 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 the Prism approach 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, whereas 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 the basic Prism and PrismTCS. PrismTCS does not need to 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.

3.2 Random Prism Classifier

The basic Random Prism architecture is illustrated in Figure 2. Random Prism uses the R-PrismTCS base classifier, which is outlined below in this Section. The prefix *R* denotes the random component of the base classifier which is Ho's [15] random feature subset selection. Random Prism uses a further random component: Breiman's bagging [6]. Both random components have been chosen in order to make Random Prism more robust towards noise in the training and test data. A further facility that has been implemented in the R-PrismTCS base classifiers, in order to improve their tolerance to noise, is J-pruning [4], a rule pre-pruning facility. Random Prism induces multiple R-PrismTCS classifiers and all of their rule sets are aggregated and used to classify new data instances employing a weighted majority voting. As the Random Prism classifier is inspired by the principles of the RF ensemble learner, 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.

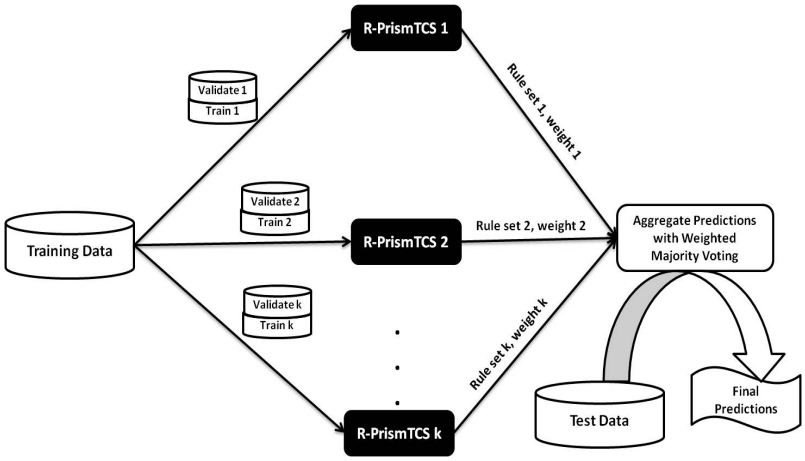

Fig. 2 The Random Prism architecture comprising Bagging, R-PrismTCS base classifiers and weighted majority voting.

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 RDF's random feature subset selection, RF's bagging and PrismTCS as base classifier.

Using Prism as a 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 produce less overfitting 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 [27]. A good computational efficiency is needed, because 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, a sample D_i is created, where i is the *ith* classifier. For the creation of D_i bagging and random sampling with replacement [6] are used. 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 they 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 use in Random Prism based on PrismTCS's pseudo code in Section 3.1. 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*, with the initial R 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 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);
}
```

Please note that the Random Prism pseudo code above, creates not only a set of classifiers 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 [29], 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 that 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 percentage of classifiers to be used for prediction. Random Prism will always select those classifiers with the highest weights.

For example consider the 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 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 to further generalise the base classifiers. J-pruning is based on the J-measure. According to Smyth and Goodman [23], the average information content of a rule of the form $IF\ Y = y\ THEN\ X = x$ can be quantified by the following equation:

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

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:

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

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.

4 Evaluation of Random Prism Classification

Random Prism's performance has been evaluated in terms of classification accuracy on several datasets highlighted in Section 4.1 and in terms of its tolerance to noise in Section 4.2.

4.1 Random Prism's Classification Accuracy

Random Prism's classification accuracy has been evaluated on 15 different datasets retrieved from the UCI data repository [2] [24]. 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 3.2 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 the Random Prism classifier and the accuracy achieved using a single PrismTCS classifier. 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, i.e. for datasets 'car evaluation', 'segmentation', 'soybean' and 'contact lenses', it can be seen

Table 2 Accuracy of Random Prism compared with PrismTCS [24].

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

that the accuracies for 'car evaluation' and 'contact lenses' are 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.

4.2 Evaluation of Random Prism's Noise Tolerance

Noise in an unavoidable problem in many domains of data mining, hence it is important that data mining and thus classification rule induction algorithms exhibit tolerance to noisy datasets.

The evaluation of Random Prism's noise tolerance has been conducted on 3 datasets from the UCI repository, the vote, contact lenses and breast cancer datasets. Several versions of the training and test datasets have been generated introducing different levels of noise in both the test and the training set. For example if the noise level was 10% then for each instance in the test or training set every attribute, including the classification, is replaced by a noise value with a probability of 10%. A noise value is any valid value for this attribute including its original value.

For each level of noise in the training set (0%, 10%, 20%, 30%, 40% and 50%) a Random Prism classifier and a PrismTCS classifier was induced using the same parameter setting used for the experiments described in Section 4.1. This classifier was then used to classify the test sets with different noise levels (0%, 10%, 20%, 30%, 40%, 50%, 60%, 70% and 80%). The number of correct classifications for both Random Prism and PrismTCS has been counted and is compared for all three datasets in Figures 3, 4 and 5.

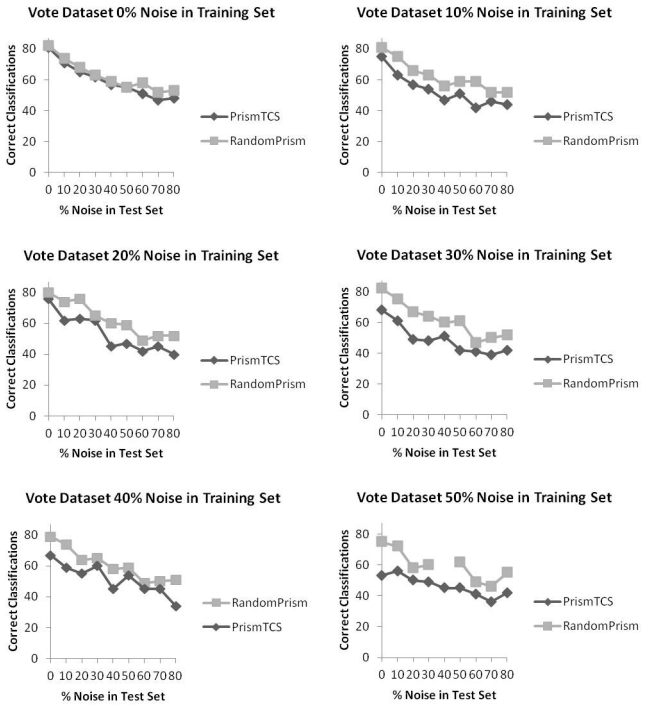

Fig. 3 Random Prism's behaviour when introducing noise into the 'vote' training and test sets.

Figure 3 compares the number of correct classifications of Random Prism and PrismTCS for increasing levels of noise in the training and test datasets of the 'vote' dataset. It can be observed that Random Prism outperforms PrismTCS in almost all cases.

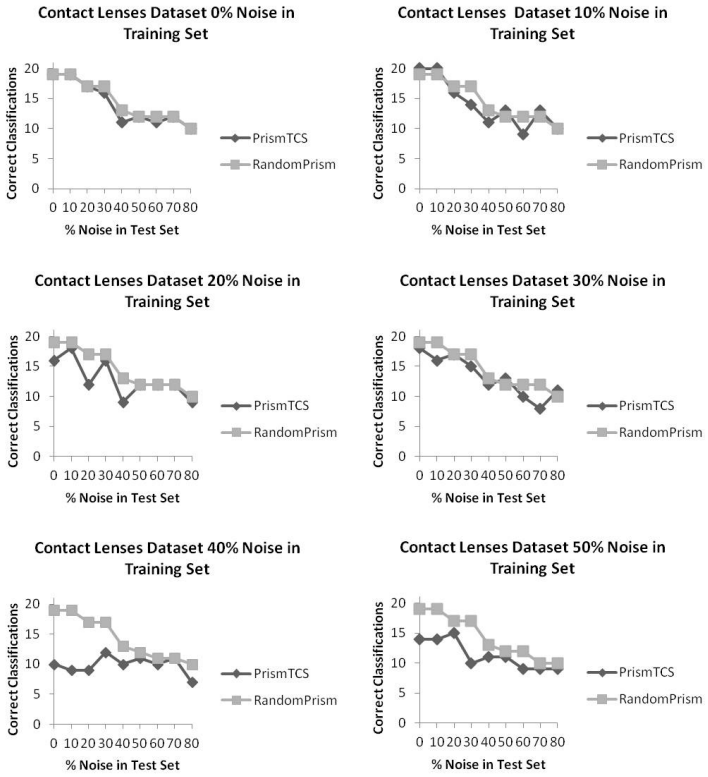

Fig. 4 Random Prism's behaviour when introducing noise into the 'contact lenses' training and test sets.

Figure 4 compares the number of correct classifications of Random Prism and PrismTCS for increasing levels of noise in the training and test datasets of the 'contact lenses' dataset. Again, it can be observed that Random Prism outperforms PrismTCS in almost all cases.

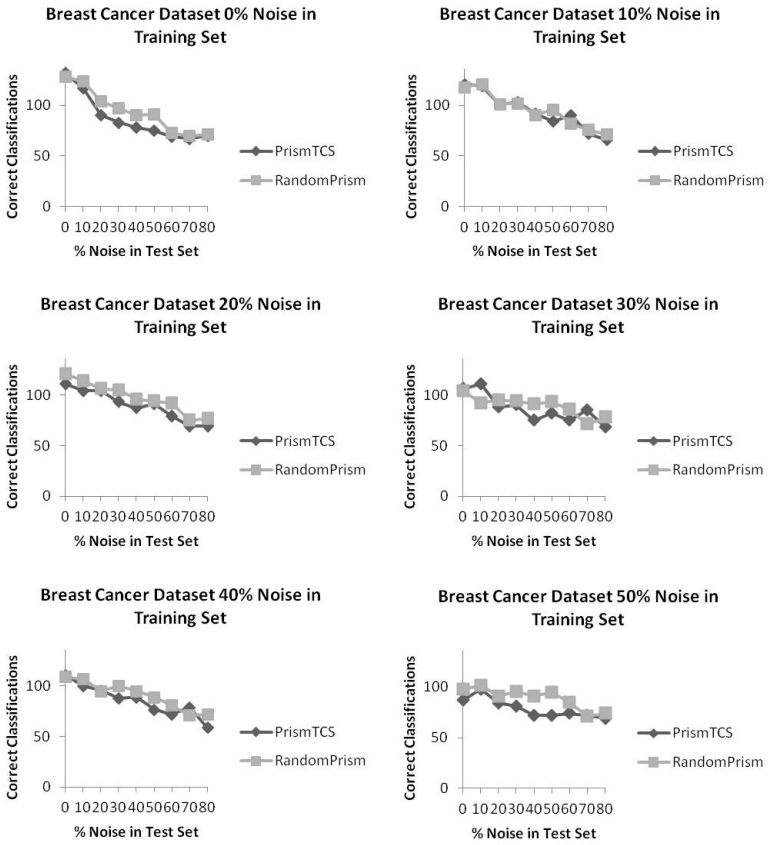

Fig. 5 Random Prism's behaviour when introducing noise into the 'breast cancer' training and test sets.

Figure 5 compares the number of correct classifications of Random Prism and PrismTCS for increasing levels of noise in the training and test datasets of the 'breast cancer' dataset. Once again, it can be observed that Random Prism outperforms PrismTCS in almost all cases.

In general it can be observed that for both Random Prism and PrismTCS the number of correct classifications decreases with an increasing noise level in the training and test set. However, as expected, in most cases Random Prism showed a higher tolerance to noise compared with PrismTCS.

5 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.

5.1 Parallel Random Prism Classifier

Random Prism, like any other ensemble learner, has a higher demand on CPU time compared with its single classifier version. Table 3 lists the runtimes of PrismTCS and Random Prism for the evaluation experiments outlined in Section 4.1. 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 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

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 greater than for PrismTCS as Random Prism induces 100 base classifiers, however the runtimes are much shorter than expected. The reason 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 [19] 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 mapping function to workers (mappers) in the MapReduce cluster. Each mapper then applies the mapping function on its data. The output of the mapping function (a key-value pair) is then grouped and combined by a second kind of worker, the reducer, using a user defined reducer function.

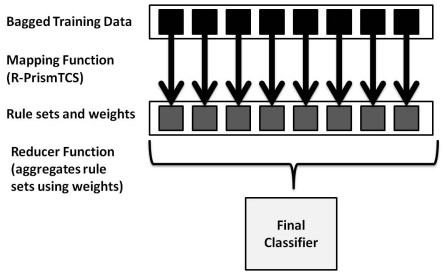

Fig. 6 Random Prism parllelised using the MapReduce framework.

Figure 6 highlights the parallelisation of Random Prism using MapReduce. For Random Prism, the MapReduce model will be used to distribute the induction of the R-PrismTCS base classifiers using map functions. The individual R-PrismTCS classifiers are then combined using a reducer function to create a final Random Prism classifier. Executing each mapping function on a different computer in the MapReduce cluster results in the computational expensive induction of R-PrismTCS based classifiers being distributed over a whole cluster of workstations. An open source implementation of the MapReduce model called Hadoop is available [1].

5.2 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.

5.3 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].

6 Conclusions

This work presents the Random Prism ensemble classifier based on the Prism family of algorithms as the 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 by the Prism family of algorithms, the Random Decision Forests and the Random Forests approaches. Random Prism uses the PrismTCS classifier as base classifier with some modifications called R-PrismTCS. The modifications were made 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, and 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. Random Prism has also been evaluated in terms of its tolerance to noisy training and test datasets in three cases from the UCI repository. In most cases observed, Random Prism achieved a higher classification accuracy compared with PrismTCS.

Ongoing work on Random Prism comprises the development of a parallel version in order to make Random Prism scale better on large datasets. For this the MapReduce framework for utilising a computer cluster is considered. Furthermore a more intelligent voting system taking different accuracy levels of the classifier for different classifications into account is planned. Additionally, a variety of Random Prism versions are planned, comprising different Prism classifiers as base classifiers.

Author Biographies

Dr Frederic Stahl

Frederic Stahl joined the University of Reading , UK in December 2012 where he is a lecturer. Prior to joining Reading he worked as a lecturer at Bournemouth University and as Senior Research Associate at the University of Portsmouth. He obtained his PhD from the University of Portsmouth in 2006 with the thesis title "Parallel Rule Induction". He remained an active researcher in the field of "Big Data" analytics, in particular in parallel data mining approaches. Besides parallel and distributed data mining, Dr Stahl's research interests are also in data stream mining, data mining in resource constraint environments, machine learning and artificial intelligence. Since 2011 Dr Stahl is a technical program committee member of the annual SGAI "International Conference on Artificial Intelligence". He is also author and co-author of over 24 technical peer reviewed publications.

Prof. Max Bramer

Professor Max Bramer is Emeritus Professor of Information Technology at the University of Portsmouth, UK. He has been an active researcher into different areas of Artificial Intelligence since the 1970s (PhD awarded 1977). In recent years he has focused particularly on Knowledge Discovery and Data Mining and is the author of a popular textbook on 'Principles of Data Mining'. Professor Bramer is the Chair of the Specialist Group on Artificial Intelligence of the British Computer Society and Vice-Chair of the Technical Committee on Artificial Intelligence of the International Federation for Information Processing. He was a member of the founding steering committee for the IEEE

ICDM series of data mining conferences. He is the author, co-author or editor of over 200 technical publications.

References

1. Hadoop, http://hadoop.apache.org/mapreduce/ 2012.
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. J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, December 2007.
17. 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.
18. Leandro L. Minku and Xin Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE Trans. on Knowl. and Data Eng.*, 24(4):619–633, April 2012.
19. 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.
20. Foster Provost. Distributed data mining: Scaling up and beyond. In *Advances in Distributed and Parallel Knowledge Discovery*, pages 3–27. MIT Press, 2000.

21. R J Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
22. Ross J Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
23. 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.
24. Frederic Stahl and Max Bramer. Random prism: An alternative to random forests. In Max Bramer, Miltos Petridis, and Lars Nolle, editors, *Research and Development in Intelligent Systems XXVIII*, pages 5–18. Springer London, 2011.
25. Frederic Stahl and Max Bramer. Computationally efficient induction of classification rules with the pmcri and j-pmcri frameworks. *Knowledge-Based Systems*, 35(0):49 – 63, 2012.
26. Frederic Stahl and Max Bramer. Jmax-pruning: A facility for the information theoretic pruning of modular classification rules. *Knowledge-Based Systems*, 29(0):12 – 19, 2012.
27. 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.
28. Frederic Stahl, Mohamed Gaber, P. Aldridge, D. May, H. Liu, Max ramer, and P. Yu. Homogeneous and heterogeneous distributed classification for pocket data mining. In A. Hameurlain, J. Kung, and R. Wagner, editors, *Transactions on large-scale data and knowledge-centered systems V*, volume 5 of *Lecture notes in computer sciences*. Springer, Berlin, 2012.
29. 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, 2011. Springer.
30. Frederic Stahl, MohamedMedhat Gaber, and ManuelMartin Salvador. erules: A modular adaptive classification rule learning algorithm for data streams. In Max Bramer and Miltos Petridis, editors, *Research and Development in Intelligent Systems XXIX*, pages 65–78. Springer London, 2012.
31. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.
32. I H Witten and F Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
33. Indre Zliobaite. Three data partitioning strategies for building local classifiers. In G. Valentini, M. Re, and O. Okun, editors, *Ensembles in Machine Learning Applications*. Springer, Heidelberg, 2011.