Random Rule Sets – Combining Random Covering with the Random Subspace Method

Tony Lindgren

***Abstract*—Ensembles of classifiers has proven itself to be among the best methods for creating highly accurate prediction models. In this paper we combine the random coverage method which facilitates additional diversity when inducing rules using the covering algorithm, with the random subspace selection method which has been used successfully by for example the random forest algorithm. We compare three different covering methods with the random forest algorithm; 1st using random subspace selection and random covering; 2nd using bagging and random subspace selection and 3rd using bagging, random subspace selection and random covering. The results show that all three covering algorithms do perform better than the random forest algorithm. The covering algorithm using random subspace selection and random covering performs best of all methods. The results are not significant according to adjusted p-values but for the unadjusted p-value, indicating that the novel method introduced in this paper warrants further attention.**

***Index Terms*—Ensemble of classifiers, random covering, random subspace selection, diversity.**

I. Introduction

Induction of rules are a formidable way of creating prediction models, the rules induced are easy to interpret and usually have a good predictive power [1]. When using ensemble methods, it is enough to have an inductive algorithm that have a predictive power better than random and able to create diverse models to create a powerful predictive ensemble using unweighted majority vote of all models that is part of the ensemble, for details see [2]. When considering ensemble of classifiers, it is clear that it is not the strength of each individual learner that determines the overall performance but rather the combination of individual *learners*, their *diversity* and the *number* of learners used in the ensemble. These three factors can be manipulated for improving ensembles. In this paper we mainly focus on how we can facilitate diversity in the inductive algorithm used. We present a modified rule induction method based on the covering algorithm. The method we introduce in this paper has its origins in the algorithm presented here [3], where we first introduced *random coverage of examples*, i.e. the amount of examples not remove when creating a rule, which is user defined via a hyperparameter.

The paper starts with a review of the existing work on key factors for ensembles. After that we will present our proposed method for random rule set induction and relate our method to the key factors introduced earlier. We will then present the experimental setting were we will compare our novel method, in different settings, with an established method, in this case the random forest algorithm. We then present the results of the experiment and finish off the paper with a discussion and reason about possible future work.

Manuscript received October 6, 2017; January 8, 2018. This work has been funded by Scania CV AB and the Vinnova program for Strategic Vehicle Research and Innovation (FFI)-Transport Efficiency.

Tony Lindgren is with the Department of Computer and Systems Sciences at Stockholm University, Nodhuset, Borgarfjordsgatan 12, S-164 07 Kista, Sweden (e-mail: tony@dsv.su.se).

II. Key Factors in Ensemble Learning

As mentioned in the introduction there exists different factors that can be identified and adjusted to improve the performance of the ensemble as a whole. Below we will look at these factors in isolation but also discuss how they relate and influence each other.

But before we delve into these matter we first define our setting and what we mean by an ensemble. Here we follow the definitions presented by [4]. In the standard supervised learning setting, where *supervised* refers to that each example in the training set has an associated class or a numerical value. In the former setting we have a *classification* problem and in the latter we have a *regression* problem. In our case we will only consider classification problems. The training examples have the form $(x_1, y_1),\ldots,(x_m, y_m)$ were $y = f(x)$ is an unknown function that we are aiming to learn. When performing classification, one typical setting is the binary classification problem, where $y \in \{\text{pos}, \text{neg}\}$, we will refer to this set of classes as C, note that C is not restricted to contain only two class labels. The example x_i is typically a vector $\langle x_{i,1}, x_{i,2}, \ldots, x_{i,n}\rangle$, were the components can be of different types, all describing some properties, or *features*, of the example. The data type could either be a categorical value, i.e. $x_{i,j} \in \{\text{red}, \text{green blue}\}$, or a numerical value of some sort, i.e. $x_{i,j} \in \{\mathbb{R}, \mathbb{Z},...\}$.

These are the most common data types but data can come in other shapes, for example as histograms, i.e. where components of the vector are them self a vector or a matrix, these types of data is rather unusual but if one is interested in utilizing these sub-structures, special algorithms can be used that consider the histogram structure while learning the function y, see for example [5].

In our notation $x_{i,j}$ j refer to the example and i to a feature, hence the a combination of i,j will point out a feature for a particular training example. The set of labeled training examples we denote X. A classification learning setting thus involves X which are input to a learning algorithm which in turn produces a classifier, consisting of a hypothesis h about the function $y = f(x)$. In an ensemble setting we will have S as input to one or more (hyperparameter ensemble size) learners

doi: 10.18178/ijmlc.2018.8.1.655

which will produce classifiers denoted by $h_1, \ldots, h_S$. One of the key issues for ensembles is how to combine the hypothesis h_i into one classifier. How this has been done previously, is what we will look at now.

A. Voting Scheme

An ensemble is constructed by a training set X and by setting an ensemble size hyperperameter to S we will construct S hypothesis, $h_1, \ldots, h_S$, (also referred to as models or classifiers). As mentioned earlier these hypotheses must be diverse, we will get into this issue in detail later on, so for now let us use an intuitive description of diversity with the meaning that we *want different classifiers to make different faults*. The algorithm used to create a hypothesis can have different types of outputs given what algorithm has been used. According to [6] there exist different levels of output from the classifiers (or hypothesis):

1) oracle: Given a **x** to classify and a classifier h_i we get the result of the prediction as right or wrong.

$$y_{i,j} = \begin{cases} 1, \text{if } h_i(x) = y_j \\ 0, \text{otherwise} \end{cases}$$

2) abstract: Given a **x** to classify and a classifier h_i we get the result of the prediction as a class label $c \in C$.
3) rank: Given a **x** to classify and a classier h_i we get the result of the prediction as alternative class labels ranked according to their plausibility, $c \in C$.
4) measurement: Given a **x** to classify and a classier h_i we get the result of the prediction as the probabilities for each class label $c \in C$.

Given these different types of outputs from the classifiers, we can identify different methods for combining the classifiers. Majority voting is one of the most common method for combining classifiers.

$$\sum_{i=1}^{S} d_{i,c_{maj}}(X) = \max_{j=1}^{c} \sum_{i=1}^{S} d_{i,j}(X)$$

Here S is (as earlier) the number of classifiers we use in the ensemble. Majority voting acts on the output labels from the classifiers, hence it operates on the abstract level, $d_{i,j}$ is either 0 or 1 depending if the classifier i outputs j or not. The ensemble chooses c_{maj} to be the class which receive the largest vote and hence outputs this as a prediction for example **x**. For a thorough an in-depth analysis of majority voting, see [7]. By utilizing some measure of quality of classifiers, weights can be added to the voting scheme to create a weighted majority voting scheme, it has been shown [8], that the optimal weights are set is to use the follow equation:

$$w_i = \log \frac{acc_i}{1 - acc_i}$$

where acc_i is the accuracy for the i:th classifier. An obvious note here is that the accuracies should be calculated from data not used when building the classifiers. The equation for weighted voting is:

$$\sum_{i=1}^{S} w_i d_{i,c_{maj}}(X) = \max_{j=1}^{c} \sum_{i=1}^{S} w_i d_{i,j}(X)$$

Stacking [9] is one method that utilizes meta-information from the base classifiers to train a model for combining the classifiers. It is apart from the different voting methods one of the most common method for combining different base learners. Other methods exist, for example one could simply sum all fired rules coverages, i.e. the number of covered training examples, and use the most frequent class as the prediction. This would correspond to a method on rank-level.

B. Ensemble Size

In Condorcet's jury theorem [10] he claims that, adding more members to the jury will increase the probability that the majority's decision is correct. This claim is valid given a few assumptions, each member of the jury should vote better than random, that they vote independently of each other and that the group of jury members face a decision problem, with an outcome that is either correct or incorrect. This theorem will when the jury size $n \to \text{inf}$, $p_n \to 1$ where p is the probability for voting correct. This theorem has later been extended to ensemble learning where jury member is swapped for classifiers and the outcome prediction of classes given an unseen example, see [2], [11].

Even though it has been shown in theory that more voters, or classifiers is better, there has recently been empirical observations that the predictive performance at some point do not gain much from added classifiers. In the work of [12] the authors investigated how many trees is necessary when using a random forest. They investigated 29 datasets and varied the ensemble size using a base of two, $S = 2^j$, j = 1, 2, …, 12 Hence the different ensemble sizes were: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 and 4096. In short they came to the conclusion that using larger number of trees than 128 is not worthwhile.

The result from [13], [14] are not equally clear cut and they indicate that different machine learning problems need different ensemble sizes. Both papers investigate how to dynamically create a dynamic stopping criteria to decide when to stop adding more classifiers to an ensemble. In the former paper they achieve this by investigating the out-of-bag error of the ensemble and in the latter paper they investigate how stable the predictions from an ensemble are and base their decision on this information.

C. Diversity and Individual Learner

One thing that is clear, if we created an ensemble with many identical classifiers the result would be identical to using just one of these classifiers. For the ensemble setting to work we need diversity. The article [13] is recommended for an in-depth investigation of different measures of diversity, here we will discuss diversity more informally.

From our initial observation that identical classifiers are not diverse we now Investigate different methods that have been used for facilitating diversity, the different methods can be classified as below:

1) use different base classifiers
2) use different feature sets
3) use different data subsets

We will now give examples of methods that facilitate diversity for each category, we will not try to define the meaning of diversity other than that these different examples will hopefully give a better intuitive understanding of the

concept. A paper that discusses these categories and more in-depth is [15].

1) Different base classifiers

Using different base classifiers with the same data is a perfectly good way of creating ensembles, it is of course hard to know in advance how, for example an SVM model will be behave differently from a decision tree. Hence different learning algorithms will give rise to different models, the error of the models can be decomposed into their bias and variance. Where bias measures how far of the model are from the correct value and variance measures how predictions for a given example varies with different realizations of the model (different dataset used for creating the model). The error of a model for a regression problem can be defined as follows:

$$\mathrm{Err}(x) = (E[h(x) - y])^2 + E\,[h(x)\text{-}E[h(x)])^2] + \sigma_e^2$$

$$\mathrm{Err}(x) = \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{IrreducibleError}$$

Above we follow the definitions of [16]. Pedro Domingos provide a unified bias-variance decomposition for the 1-0 loss and squared loss function to be used in the classification setting in [17], he also conducted experiments where he compared Decision Tree (C4.5), AdaBoost (with decision trees) and K-Nearest Neighbor. In this work Domingo's also put forward a hypothesis '...that higher-variance algorithms (or settings) may be better suited to classification (zero-one loss) than regression (squared loss)'. Using knowledge about different learning algorithms and/or impact of hyperparameters settings on the final model is a viable way of creating ensembles. But this tack has, as of yet, not been a popular way of creating ensembles.

2) Different feature sets

The random subspace method [18] which aims at creating diverse models by selecting the best feature from a randomly selected subset of features. In their paper they use the random subspace method for constructing a decision forest (ensemble) consisting of decision trees but the general method can and have been applied to other learners as well see [19, 20]. The algorithm for random subspace is shown in Algorithm 1. Input to the algorithm is the training examples (X), size of the ensemble (S), size of subspace (d) and the feature set (D). The algorithm selects randomly selects a set of features with replacement d_{set} of size S. The tree induction function is then called with this feature set d_{set} and the training examples X output is a model h_{temp} which are added to the ensemble of models H. The algorithm continues from the top until enough models have been created.

Algorithm 1 – Random subspace method
Inputs: *TrainingExamples(X), EnsembleSize(S), FeatureSet(D), SubspaceSize(d)*
Outputs: *EnsembleOfModels(H)*
while S ≠ ∅ **do**
  $d_{set} = \emptyset$
  while $d \neq \emptyset$ **do**
    $d_{set} \leftarrow d_{set} \cup$ select $d \in D$ with replacement
    $d = d - 1$
  end while
  $h_{temp} \leftarrow InduceTree(X, d_{set})$
  $H \leftarrow H \cup h_{temp}$
  $S = S - 1$
end while
return H

3) Different data subsets

The most popular methods for creating diversity is to alter the input data in different ways. One of the more popular methods is bagging (or bootstrap aggregating) proposed by [21]. The methods create different training sets which in turn are used to produce $h_1,\ldots, h_s$ models. Each dataset X_i are created by random sampling with replacement of examples from training examples X. Hence each training set will probably differ from each other w.r.t. composition of examples. When creating these bagged trainings sets it is expected that 63.2 % of the examples are unique examples from X and hence the rest duplicates. AdaBoost [22] is another popular method for creating different data subsets. Here each example has a weight w_i associated with it, initially each example has equal weights $w_i = 1 = m$ where m denotes the number of training examples. The following steps are then repeated according to the ensemble size S.

For l ∈ 1, …, S:

1) Train learner on X_t (X with associated weights W)
2) h_l: $X \rightarrow$ {pos, neg} with error ε_t
3) Choose: $w_t = ½ \ln (1 - \varepsilon_t / \varepsilon_t)$
4) Update: $X_{t+1}(i) = \frac{X_t(i)}{Z_t} \times \begin{cases} e^{-wt} if\ h_t(x_i) = y_j \\ e^{wt} if\ h_t(x_i) \neq y_j \end{cases}$

Where Z_t is a normalization factor to make sure that X_{t+1} will be a distribution. In each iteration the weights of the erroneous predicted examples weights are increased, hence making the learning in the next round to focus extra on these examples.

One difference between AdaBoost and for example bagging and the random subspace method is that it is serial in its nature, as input from an earlier stage is needed for the next stage, thus making it harder to parallelize the algorithm.

One of the most popular ensemble methods is random forest [23] which combine the random subset method together with bagging, this combination has proven very efficient and effective.

III. Using Ensembles of Rule Sets

There has been some work in the area of utilizing rule sets as base models for ensembles, as well as using rule set for combining base learners. In the paper [24] by Jerzy Stefanowski he investigates how to use three different ensemble methods with rule induction. The three methods are bagging, n^2 and a stacking type of meta-learning method. He compared the performance of the three methods on a few common datasets and his conclusions is that the n^2 method seems to performed best. In their paper [25] Petr Savicky and Johannes Fürnkranz uses rule induction (ripper) as base learner together with different combining learners in a stacking approach. The investigate three different meta-learning algorithms and their performance: decision tree induction (c4.5), rule induction (ripper) and a nearest neighbor classifier (k-nn). Their results show that the nearest

neighbor classifier sometimes did drastically improve performance as compared to using unweighted voting on the base classifier models while the other two methods did not.

In the paper [3], Tony Lindgren introduce a new parameter for the covering algorithm which allows the user to specify how many of the covered examples, by the induced rule, which should not be removed from the set of examples to be covered. Hence these examples must then be covered again, which lead to diverse sets of rules. In his paper he concludes that this modification indeed introduce diversity in rule sets. But using the method in combination with bagging did not improve performance, which was expected, instead it worked best on its own. The novel method we explore in this paper uses the new parameter in a random subset selection setting, which we will describe in detail in the next section.

IV. Novel Method

The novel method we introduce here is a combination of the earlier method presented in [3] and the random subspace method [18]. The algorithm works by randomly selecting a subset (according to a hyperparameter) of all available features in a stepwise fashion, for each condition in a rule. After a rule has been induced the new hyperparameter is then used to select randomly how many of the covered examples which should be removed from the set containing the remaining examples to be covered (refereed here on as random covering). The algorithm is more formally described in Algorithm 2.

The inputs to the algorithm is the training examples (*X*), a set of available features (*D*), size of the ensemble, i.e. number of classifiers (S), size of subspace (*d*) and finally the proportional size of examples to remove after covering (*pR*). The algorithm's main loop makes sure that we create *S* number of classifiers. The inner loop induces new rules until all examples are covered. It does this by the function *createOneRule*, which uses the current training examples (*X*) and hyperparameters *(D, d)*. The function repeatedly calls *getRandomSubSpace* with *D* and *d* this function returns a set of attributes *a*. The best condition which utilizes one attribute from *a* according to some quality criteria are then added to the rule being built. This process continues until the rule satisfies the stopping criterion or there are no examples left to cover. It then returns the covered rule (*Rule*) together with the set of examples covered by the rule (*CovEx*). The function *exToRemove* uses *CovEx* and *pR*, the function randomly selects for each example if it should remove the example from CoveredEx or not w.r.t. the hyperparameter *pR*. It then returns the (possibly) modified *CovEx* set. The examples in *CovEx* are then removed from the *X* and the inner loop is then restarted until no *X* are left.

Algorithm 2 – Random rule sets
Inputs: *TrainingEx(X), FeatureSet(D), EnsembleSize(L), SubspaceSize(d), pSizeExToRemove(pR)*
Outputs: *EnsembleOfModels(H)*

H = ∅
while S ≠ ∅ **do**
  while *X* ∉ ∅ **do**
    CovEx, Rule ← createOneRule(*X, d, D*)
    ExToRemove ← exToRemove(*CovEx, pR*)
    X ← *X* \ *ExToRemove*
    H ← *H* ∪ *Rule*
  end while
  S ← *S* – 1
end while
return *H*

function *getRandomSubSpace(d, D)*
  d_{set} = ∅
  while *d* ≠ ∅ **do**
    d_{set} ← d_{set} ∪ select *d* ∈ *D* with replacement
    d = *d* – 1
  end while
return d_{set}

function exToRemove(*CoveredEx, pToRemove)*
  for *all examples, ex* ∈ *CoveredEx* **do**
    randVal ← *randomFloat*
    if *randVal* =< *pToRemove* **then**
      CoveredEx ← *CoveredEx* \ *ex*
    end if
  end for
return *CoveredEx*
end function

function createOneRule(*X, d, D*)
  c ← ∅ //Initialize a rule with empty conditions
repeat
  for *all attributes, a* ∈ *getRandomSubSpace(d, D)* **do**
    R ← *c* = *c* ∪ *α* //Use attribute a to create a condition
    $BestC_a$ ← eval according to some quality criteria
  end for
  c ← *c* ∪ $BestC_a$
  Rule ← *c*
until *Rule satisfies a stopping criterion* ∨ *X* = ∅
  CoveredEx ← *gets all examples covered by Rule*
return *CoveredEx, Rule*
end function

V. Experiment and Results

Our empirical evaluation of the novel algorithm was set up as follows. The experiment where conducted using a 10-fold cross validation fashion, hence all results are the average values from these 10 folds. Size (number of rules) and accuracy of each algorithm was used as performance metrics. We compared the random forest algorithm together with unordered rule sets induced in three different ways. The first method utilized random subspace selection together random covering, i.e. the novel method described in Algorithm 2. The second unordered rule induction method utilized bagging together with random subset selection, i.e. the unordered rule set counterpart of random forest. The third unordered rule induction method utilized all three diversity inducing methods at once, i.e. random subspace selection, bagging and random covering. All compared algorithms were implemented using SWI-Prolog, and are available to download from http://dsv.su.se/~tony/programs.html together with the datasets used in the experiment.

All datasets were taken from the UCI repository [26], in total 28 datasets were used. The parameters that were used was the

following: ensemble size = 200, random subset size = 0.3, minimum coverage = 10, minimum margin = 0.9, random coverage = 0.2. The ensemble size was selected upon the results of [10], where they claim that larger ensembles than 128 is not necessary, so we set the size to 200, just to be sure. The random subset size of 0.3, meaning that 30% of the attributes will be considered when inducing conditions, which is a reasonable size. Minimum coverage and minimum margin are both stopping criterions, where the first stops if less than or equal to 10 examples are covered by a rule and minimum margin specifies that the margin difference between the two largest classes, should be 90% or larger to stop. Random coverage of 0.2 reflect that 20% of the covered examples are left in the training set to be covered again. The prediction of the ensemble was done using unweighted voting.

TABLE I: THE RESULTS OF THE EXPERIMENT

Dataset	Rnd. Forest		Rule1		Rule2		Rule3	
	Size	Accuracy	Size	Accuracy	Size	Accuracy	Size	Accuracy
Acute Diagnosis 1	828.2	1.000	1794.2	1.000	916.2	1.000	1774.5	1.000
Acute Diagnosis 2	661.1	1.000	1531.6	1.000	752.2	1.000	1525.1	1.000
Adult	16841.7	0.760	31363.6	0.757	18530.2	**0.772**	1945.6	0.760
Arrhythmia	1136.5	**0.629**	6222.2	0.613	4210.1	0.598	5532.9	0.604
Balance scale	17988.8	0.805	13569.5	0.856	8602.7	0.872	11819.1	**0.880**
Breast cancer Wisconsin	777.6	0.966	3295.2	0.964	1710.2	0.966	2892.2	0.961
Bupa liver disorder	1024.7	0.597	8488.5	**0.739**	5142.6	0.728	7064.2	0.730
Cleveland heart disease	968.9	0.547	7405.0	**0.593**	4991.1	0.563	6655.6	0.580
Climate model failures	430.8	0.915	3084.4	**0.941**	1658.0	0.937	2690.1	0.924
Crx	643.1	0.801	4395.6	0.865	2340.6	0.850	3940.2	0.865
Forest types	957.7	0.879	3594.7	0.867	1870.7	0.871	3424.0	0.867
Glass	679.6	0.909	1565.2	**0.945**	803.2	0.936	14443.7	0.936
Haberman	818.0	**0.735**	6549.2	0.719	4464.7	0.699	5940.8	0.712
Iris	903.9	0.940	1894.0	0.927	929.0	0.933	1816.3	0.940
Image segmentation	2486.3	**0.919**	4502.8	0.895	2541.1	0.785	4121.0	0.895
Ionosphere	1150.0	**0.926**	3191.8	0.894	1720.5	0.906	2762.1	0.861
Magic04	786.0	0.742	47019.7	0.863	22727.5	**0.865**	35615.9	0.861
Mushroom	417.3	0.938	4080.3	0.958	1275.7	**0.960**	4103.7	0.959
Pen digits	7605.0	0.953	24480.5	**0.981**	14948.6	0.976	23036.5	0.977
Pima Indians	915.8	0.703	13762.2	0.750	7843.7	**0.763**	11019.1	0.754
Sensor readings	1250.8	0.926	7892.2	**0.941**	3512.9	0.865	7523.8	0.936
Shuttle	2518.1	0.939	3402.9	**0.957**	1779.9	0.896	3289.5	0.950
Sonar	1150.3	0.769	2832.0	0.870	1563.1	0.861	2441.9	0.870
Spam base	761.0	0.892	9078.5	0.906	4754.2	**0.910**	8237.4	0.905
Spectral flare	482.2	0.728	3948.8	**0.891**	2187.4	0.862	3390.0	0.857
Thyroid	843.4	0.944	2144.5	0.944	1162.9	0.944	2017.7	0.944
Transfusion	462.8	0.762	7185.3	0.782	4502.7	0.778	6449.3	**0.783**
Wine	1115.0	0.972	2046.5	0.972	1099.9	**0.983**	1927.4	0.972
Average values	2378.7	0,843	8225.8	0,871	4590.8	0,860	6692.8	0.867
Friedman rank (p-Value: 0.12775)		2.964		2.160		2.446		2.429

Results of the Experiment

In Table I the results of the experiment are shown, the first column denotes the name of the dataset for that row, then each pair of columns denote the size and accuracy of the respective method over the 10 folds. First comes the results from the random forest, secondly the covering method (Rule1) using random subspace and random covering, then the covering method (Rule2) bagging and random subspace and finally (Rule3) uses bagging, random subspace and random covering. If there exists a single best method with highest accuracy for a dataset that accuracy is written using **bold** letters. If one counts the number of best results we see that the random forest algorithm collects 4 wins, Rule1 8 wins, Rule2 5 wins and Rule3 2 wins. Given this the Rule1 method seems to perform best, and Rule3 worst, with Rule5 and random forest on par. Using the statistical test of [27] we can rank the algorithms using the Friedman rank, the ordering changes a bit, see bottom row of Table I, with (still) Rule1 best, then Rule2, Rule3 and last random forest. Note that this value is not statistically significant, having a p-value of 0.128. If we look at the size of the ensembles, the random forest produces by far the smallest number of rules in its ensemble, which is expected due to how the tree induction algorithm reclusively disjunctively-divides the feature space. Somewhat surprising is that Rule1 produces the largest rules size in its ensemble, here Rule3 was expected to create the largest rules space, as it utilizes all the different methods for creating more diverse rules sets. But it at least comes before Rule2 and is not far from Rule1. To further investigate the strengths of the different algorithms we conducted paired statistical significant tests, between all algorithms, the results are shown in Table II. The first column denotes which pairs we are considering, the second column show the unadjusted p-value, and the following columns show the results for the adjusted p-values for Nemenyi's, Holm's, Shaeffer's and Bergmann's procedure. From the results we can cannot reject the null hypothesis, i.e. that there is no statistically difference according to the p-value of 0.05 between any of the models. The unadjusted p-value for random forest vs Rule1 is below the threshold but not the adjusted values, which all are well above the threshold of 0.05.

TABLE II: THE P-VALUES FROM PAIRWISE STATISTICAL TESTS

i	hypothesis	Un. p	P_{Neme}	P_{Holm}	P_{Shaf}	P_{Berg}
1	R.F. vs .R1	0.0199	0.1192	0.1192	0.1192	0.1192
2	R.F. vs .R3	0.1205	0.7230	0.6025	0.3615	0.3615
3	R.F. vs .R2	0.1334	0.8003	0.6025	0.4002	0.3615
4	R1 vs .R2	0.4076	2.4458	1.2229	1.2229	1.2229
5	R1 vs .R3	0.4376	2.6253	1.2229	1.2229	1.2229
6	R2 vs .R3	0.9587	5.7523	1.2229	1.2229	1.2229

VI. DISCUSSION AND CONCLUSION

We have in this paper introduced a novel method (Rule1) by combining the random subspace section with random coverage to facilitate diversity in rules using the covering algorithm for inducing unordered rule sets. We did compare this method with the random forest algorithm and its counterpart for unordered rule induction (Rule2) and an algorithm for combining all three diversity facilitating (Rule3) methods in one algorithm, bagging, random subset selection and random covering. Even though the result were not statistically significant at the p value of 0.05, the results are promising having a Friedman rank p-value at just below 0.13. It seems that the proposed method Rule1 warrants further attention. One venue forward would be to set up an experiment with more datasets to be able to answer the question if there indeed exist differences between the methods or if it is random fluctuations. It would also be interesting to investigate if one can device a better method than just randomly selecting examples to keep in the random covering method. One idea could be to keep examples which have belonged to the minority class of the rule and hence face the danger of (erroneous) being classified as belonging to the majority class. Yet another type of tack would be to introduce weights to examples, which then can be adjusted when we have covered an example (similar to boosting).

REFERENCES

[1] J. Fürnkranz, D. Gamberger, and N. Lavrač, *Foundations of Rule Learning*, Springer verlag, 2012.

[2] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 993-1001, October 1990.

[3] T. Lindgren, "Randomized separate and conquer rule induction," in *Proc. the International Conference on Compute and Data analysis (ICCDA)*, 2017, pp. 207-214.

[4] T. G. Dietterich, "Ensemble methods in machine learning," in *Proc. the First International Workshop on Multiple Classifier Systems (MCS)*, pp. 1-15, 2000.

[5] R. Gurung, T. Lindgren, and H. Boström, "Learning decision trees from histogram data using multiple subsets of bins," in *Proc. the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference*, 2016, pp. 430-435.

[6] L. I. Kuncheva, *Combining Pattern Classiers: Methods and Algorithms*, Wiley-Interscience, 2004.

[7] D. Ruta and B. Gabrys, "A theoretical analysis of the limits of majority voting errors for multiple classifier systems," *Pattern Analysis and Applications*, vol. 5, no. 4, pp. 333-350, 2002.

[8] S. Nitzan and J. Paroush, "Optimal decision rules in uncertain dichotomous choice situations," *International Economic Review*, vol. 23, no. 2, pp. 289-297, 1982.

[9] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241-259, 1992.

[10] R. B. Myerson, "Extended poisson games and the condorcet jury theorem," *Games and Economic Behavior*, vol. 25, pp. 111-131, 1997.

[11] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 1-39, 2010.

[12] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *Proc. the 8th International Conference on Machine Learning and Datamining in Pattern Recognition*, 2012, pp. 154-168.

[13] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "A comparison of decision tree ensemble creation Techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 173-180, January 2007.

[14] Daniel Hernandez-Lobato, Gonzalo Martinez-Munoz and Alberto Suarez, "How large should ensembles of classifiers be?", *Pattern Recognition*, vol. 46, no. 5, pp. 1323-1336, May 2013.

[15] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, no. 2, pp. 181-207, 2003.

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data mining, Inference and, Prediction*, Springer Series in Statistics, Springer Ney York Inc., New York, NY, USA, 2001.

[17] P. Domingos, "A unified bias-variance decomposition and its applications," in *Proc. the 17th International Conference on Machine Learning*, 2000, pp. 231-238.

[18] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, August 1998.

[19] C. Padilha, A. D. Neto, and J. D. Melo, "RSGALS-SVM: Random Subspace Method Applied to a LS-SVM Ensemble Optimized by genetic algorithm," in *Proc. 13th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, 2012, pp. 253-260.

[20] T. K. Ho, "Nearest neighbors in random subspaces," in *Proc. Advances in Pattern Recognition: Joint IAPPR International Workshops SSPR'98 and SPR'98*, 1998, pp. 640-648.

[21] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123-140, August 1996.

[22] R. E. Schapire, "A brief introduction to boosting," in *Proc. the 16th International Joint Conference on Artificial Intelligence*, 1999, vol. 2, pp. 1401-1406.

[23] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[24] J. Stefanowski, "On combined classifiers, rule induction and rough sets," *Transactions on Rough Sets VI*, New York: Springer-Verlag, 2007, pp. 329-350.

[25] P. Savicky and J. Fürnkranz, "Combining pairwise classifiers with stacking," in *Proc. the 5th International Symposium on Intelligent Data Analysis*, 2003, pp. 219-229.

[26] M. Lichman, "UCI machine learning repository," Irvine, CA: University of California, School of Information and Computer Science, 2013.

[27] S. Garca and F. Herrera, "An extension on 'statistical comparisons of classifiers over multiple data sets' for all PAIRWISE Comparisons," *Journal of Machine Learning Research*, vol. 9, pp. 2677-2694, 2008.

Tony Lindgren was born in Hägersten, Stockholm in 1974. He received his master degree in computer and system sciences in 1999. In 2006, he received his Ph.D. degree in computer and system sciences. He has worked both in academia and industry since 2008, he is the inventor of numerous patents and has a permanent position as lecturer at the department of computer and system sciences at stockholm university since 2012. His main interest is in the field of machine learning, artificial intelligence and constraint programming.