Random Rule Sets: Combining Random Coverage with the Random Subspace Method

Tony Lindgren

***Abstract*—Ensembles of classifiers have been proven to be among the best methods to create highly accurate prediction models. In this study, 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 the random forest algorithm. We compare three different covering methods with the random forest algorithm: (1) using random subspace selection and random coverage, (2) using bagging and random subspace selection, and (3) using bagging, random subspace selection, and random coverage. The results show that all three covering algorithms perform better than the random forest algorithm. The covering algorithm using random subspace selection and random coverage performs best among all methods. The results are not significant according to adjusted *p* values but they are according to unadjusted *p* values, indicating that the novel method introduced in this study warrants further attention.**

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

I. Introduction

Induction of rules is a formidable way of creating prediction models; the rules induced are easy to interpret and usually have good predictive power [1]. When using ensemble methods, it is enough to have an inductive algorithm that has better predictive power than random and is able to create diverse models to create a powerful predictive ensemble using the unweighted majority vote of all models, which is part of the ensemble (for details, see [2]). When considering an ensemble of classifiers, it is clear that the strength of each individual learner does not determine the overall performance, rather the combination of individual *learners*, their *diversity*, and the *number* of learners used in the ensemble determine the overall perfomance. These three factors can be manipulated to improve ensembles. In this study, 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 that we introduce in this study originated from the algorithm presented here [3], where we first introduced *random coverage of examples*, i.e., the amount of examples not removed when creating a rule, which is user-defined via a hyperparameter.

The paper begins with a review of the existing work on key factors for ensembles. Following that, we present our proposed method for random rule set induction and relate our method to the key factors introduced earlier. We then present the experimental settings where we compare our novel method, in different settings, with an established method, in this case the random forest algorithm. Next, we present the results of the experiment and conclude the paper with a discussion on and reasons for 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 exist different factors that can be identified and adjusted to improve the performance of the ensemble as a whole. In the following section, we will examine these factors in isolation and also discuss how they relate to and influence each other.

However, before we delve into these matters, we first define our setting and what we mean by an ensemble. Here, we follow the definitions presented in [4]. In the standard supervised learning setting, the term *supervised* refers to the notion 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 $(x_1, y_1),\ldots, (x_m, y_m)$, where $y = f(x)$ is an unknown function that we aim to learn. When performing classification, one typical setting is the binary classification problem, where $y \in \{\text{pos, neg}\}$; 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 $\langle x_{i,1}, x_{i,2}, \ldots, x_{i,n}\rangle$, where the components can be of different types, all describing some properties, or *features*, of the example. The data type could be either a categorical value (i.e., $x_{i,j} \in \{\text{red, green blue}\}$) or a numerical value of some sort (i.e., $x_{i,j} \in \{\mathbb{R}, \mathbb{Z}, \ldots\}$).

These are the most common data types, but data can come in other shapes too, for example, as histograms, where the components of the vector are themselves a vector or a matrix; such types of data are rather unusual, but if one is interested in utilizing these substructures, special algorithms that consider the histogram structure while learning the function y can be used (see, e.g., [5]).

In our notation $x_{i,j}$, j refers to the example and i to a feature, and hence the combination of i, j will highlight a feature for a particular training example. We denote the set of labeled training examples by X. A classification learning setting thus involves X, which is input to a learning algorithm that in turn produces a classifier, comprising a hypothesis h about the

doi: 10.18178/ijmlc.2018.8.1.655

function $y = f(x)$. In an ensemble setting, we will have S as input to one or more (hyperparameter ensemble size) learners, 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 was done previously is what we will examine now.

A. Voting Scheme

An ensemble is constructed using a training set X and by setting an ensemble size hyperparameter 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; 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 based on what algorithm has been used. According to [6], there exist different levels of output from the classifiers (or hypothesis):

(1) *Oracle*. Given **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 **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 **x** to classify and a classifier h_i, we get the result of the prediction as alternative class labels ranked according to their plausibility, $c \in C$.

(4) *Measurement*. Given **x** to classify and a classifier 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 methods for combining classifiers:

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

Here, S is (as stated earlier) the number of classifiers used in the ensemble. Majority voting acts on the output labels from the classifiers, and hence it operates on the abstract level; $d_{i,j}$ is either 0 or 1, depending on whether the classifier i outputs j or not. The ensemble chooses c_{maj} to be the class that receives the largest vote and hence outputs this as a prediction, for example, **x**. For a thorough and in-depth analysis of majority voting, see [7]. 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 in [8] that the optimal weights are set to use the following equation:

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

where acc_i is the accuracy for the ith 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

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

Stacking [9] is one method that utilizes meta-information from the base classifiers to train a model to combine the classifiers. This model is, apart from the different voting methods, one of the most common methods for combining different base learners. Other methods exist; for example, one could simply sum all fired rule 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

Condorcet's jury theorem [10] 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, i.e., they should vote independently of each other, and the group of jury members faces a decision problem, with an outcome that is either correct or incorrect. This theorem will when the jury size $n \to$ inf, $p_n \to 1$, where p is the probability of correct voting. This theorem has later been extended to ensemble learning where jury members are swapped for classifiers and the outcome prediction of classes is given as an unseen example (see [2, 11]).

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

The results 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 criterion to decide when to stop adding more classifiers to an ensemble. In the former paper, this was achieved by investigating the out-of-bag error of the ensemble, and in the latter paper, the authors investigated how stable the predictions from an ensemble are and based their decision on this information.

C. Diversity and Individual Learners

One thing that is clear is that 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.

Based on our initial observation that identical classifiers are not diverse, we now investigate different methods that have been used to facilitate diversity; these different methods can be classified as follows:

(1) Use different base classifiers.

(2) Use different feature sets.

(3) Use different data subsets.

We will now present examples of methods that facilitate diversity for each category; we will not attempt to define the meaning of diversity; apart from that, these different examples will hopefully give a better intuitive understanding of the concept. A paper that discusses these categories in more detail is [15].

1) Different base classifiers

Using different base classifiers with the same data is a sound method to create ensembles; it is of course hard to predict how, for example, a SVM model will behave differently from a decision tree. Hence, different learning algorithms will produce different models; the error of the models can be decomposed into their bias and variance, where bias measures how far the model is from the correct value and variance measures how predictions for a given example vary with different realizations of the model (different datasets used for creating the model). The error of a model for a regression problem can be defined as follows:

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

$$\mathrm{Err}(x) = \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}.$$

In the above equation, we follow the definitions of [16]. Pedro Domingos provided an 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 trees (C4.5), AdaBoost (with decision trees), and *K*-nearest neighbor. In that study, Domingo also hypothesized, "…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 hyperparameter settings on the final model is a viable way of creating ensembles, but this method has, till date, not been a popular way of creating ensembles.

2) Different feature sets

The random subspace method [18] aims to create diverse models by selecting the best feature from a randomly selected subset of features. In their paper, they used the random subspace method to construct a decision forest (ensemble) consisting of decision trees, but the general method can be and has been applied to other learners as well (see [19, 20]). The algorithm for random subspace is shown in Algorithm 1. The input to the algorithm is the training examples (*X*), the size of the ensemble (*S*), the size of the subspace (*d*), and the feature set (*D*). The algorithm randomly selects a set of features with replacement d_{set} of size *S*. The tree induction function is then called by the algorithm with this feature set d_{set}, and the training example *X* output is a model h_{temp} that is 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 method to create diversity is to alter the input data in different ways. One of the more popular methods is bagging (or bootstrap aggregation) proposed in [21]. The methods create different training sets that, in turn, are used to produce $h_1, \ldots, h_s$ models. Each dataset X_i is created via random sampling with the replacement of examples from training examples *X*. Hence, each training set will probably differ from each other with respect to the composition of examples. When creating these bagged training sets, it is expected that 63.2% of the examples are unique examples from *X* and hence the rest are duplicates. AdaBoost [22] is another popular method for creating different data subsets. Here, each example has an assosciated weight w_i; 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 learners on X_t (*X* with associated weights *W*),

(2) h_l: $X \rightarrow$ {pos, neg} with error ε_t,

(3) choose $w_t = \frac{1}{2} \ln(1 - \varepsilon_t/\varepsilon_t)$,

(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 ensure that X_{t+1} will be a distribution. In each iteration, the weights of the erroneous predicted examples are increased, hence making the learning in the next round focus more on these examples.

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

One of the most popular ensemble methods is random forest [23], which combines the random subset method together with bagging; this combination has proven extremely 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 sets to combine base learners. In the study [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 performances of the three methods on a few common datasets, and concluded that the n^2 method seems to have the best performance. In [25], Petr Savicky and Johannes Fürnkranz used rule induction (ripper) as a base learner together with different combining learners in a stacking approach. They

investigated three different meta-learning algorithms and their performances: decision tree induction (c4.5), rule induction (ripper), and a nearest neighbor classifier (*k*-NN). Their results showed that the nearest neighbor classifier sometimes drastically improved the performance as compared with using unweighted voting on the base classifier models, whereas the other two methods did not.

In [3], Tony Lindgren introduced 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, are to be covered. Hence, these examples must then be covered again, which leads to diverse sets of rules. In his study, he concluded that this modification indeed introduced diversity in rule sets. However, using the method in combination with bagging did not improve the performance, which was expected; instead, the method worked best on its own. The novel method that we explore in this study uses the new parameter in a random subset selection setting, which is described in detail in the next section.

IV. Novel Method

The novel method that 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 used to randomly select the covered examples that should be removed from the set comprising the remaining examples to be covered (referred to hereon as random coverage). This algorithm is more formally described in Algorithm 2.

The inputs to the algorithm are the training examples (*X*), a set of available features (*D*), the size of the ensemble, i.e., the number of classifiers (S), the size of the subspace (*d*), and finally the proportional size of examples to remove after covering (*pR*). The algorithm's main loop ensures that we create *S* classifiers. The inner loop induces new rules until all examples are covered using the function *createOneRule*, which uses the current training examples (*X*) and hyperparameters (*D*, *d*). The function repeatedly calls *getRandomSubSpace* with *D* and *d*, and then this function returns a set of attributes *a*. The best condition that utilizes one attribute from *a* according to some quality criteria is 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* and randomly chooses in each example whether it should remove the example from CoveredEx with respect to the hyperparameter *pR*. It then returns the (possibly) modified *CovEx* set. The examples in *CovEx* are then removed from *X* and the inner loop is restarted until no *X*'s are left.

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

$H = \varnothing$
while S $\neq \varnothing$ **do**
  while $X \notin \varnothing$ **do**
    CovEx, Rule ← createOneRule(*X, d, D*)
    ExToRemove ← exToRemove(*CovEx, pR*)
    $X \leftarrow X \setminus ExToRemove$
    $H \leftarrow H \cup Rule$
  end while
  $S \leftarrow S - 1$
end while
return *H*

function *getRandomSubSpace(d, D)*
  $d_{set} = \varnothing$
  while $d \neq \varnothing$ **do**
    $d_{set} \leftarrow d_{set} \cup$ select $d \in D$ 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**
    $R \leftarrow c = c \cup \alpha$ //Use attribute a to create a condition
    $BestC_a \leftarrow$ eval according to some quality criteria
  end for
  $c \leftarrow c \cup BestC_a$
  Rule ← *c*
until *Rule satisfies a stopping criterion* ∨ $X = \varnothing$
  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 was conducted in a ten-fold cross-validation fashion, and hence all results are the average values obtained from these ten runs. The size (number of rules) and accuracy of each algorithm were 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 with random coverage (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 the random forest). The third unordered rule induction method utilized all three diversity inducing methods simultaneously: random subspace

selection, bagging, and random coverage. All the 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 obtained from the UCI repository [26]; in total, 28 datasets were used. The parameters used were as follows: ensemble size = 200, random subset size = 0.3, minimum coverage = 10, minimum margin = 0.9, and random coverage = 0.2. The ensemble size was selected based on the results of [10], where they claimed that having ensembles larger than 128 is not necessary, so we set the size to 200 to be sure. The random subset size is 0.3, meaning that 30% of the attributes will be considered when inducing conditions, which is a reasonable size. The minimum coverage and minimum margin are both stopping criteria, where the first stops if less than or equal to ten examples are covered by a rule and the minimum margin specifies that the margin difference between the two largest classes should be ≥90% to stop. Random coverage of 0.2 reflects the notion that 20% of the covered examples are left in the training set to be covered again. The ensemble was predicted using unweighted voting.

TABLE I: EXPERIMENTAL RESULTS.

Dataset	Rnd. forest		Rule 1		Rule 2		Rule 3	
	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

Table I shows the results of the experiment; the first column denotes the name of the dataset for that row, and each pair of columns denotes the size and accuracy of the respective method over the ten runs. The results from the random forest come first, followed by the covering method (Rule 1) using random subspace and random coverage and then the covering method (Rule 2) using bagging and random subspace, and finally (Rule 3) that uses bagging, random subspace, and random coverage. If there exists a single best method with high accuracy for a dataset, that accuracy is highlighted using **bold** letters. On counting the number of best results, we observe that the random forest algorithm collects four wins, Rule 1 eight wins, Rule 2 five wins, and Rule 3 two wins. Hence, the Rule 1 method seems to perform best, and Rule 3 performs worst, with Rule 5 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 the bottom row of Table I), with Rule 1 still being the best, followed by Rule 2, Rule 3, and finally 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 because of how the tree induction algorithm reclusively disjunctively divides the feature space. Somewhat surprising is that Rule 1 produces the largest rule size in its ensemble; here, Rule 3 was expected to create the largest rule space, as it utilizes all the different methods for creating more diverse rule sets, but it at least comes before Rule 2 and is not far from Rule 1. To further investigate the strengths of the different algorithms, we conducted paired statistical significance tests between all the algorithms; the results are shown in Table II. The first column

denotes the considered pairs, the second column shows the unadjusted *p* values, and the following columns show the results for the adjusted *p* values for Nemenyi's, Holm's, Shaeffer's, and Bergmann's procedure, respectively. Based on these results, we can cannot reject the null hypothesis: that there is no statistical difference according to the *p* value of 0.05 between any of the models. The unadjusted *p* value for the random forest versus Rule 1 is below the threshold but not the adjusted values, which are all 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	RF vs. R1	0.0199	0.1192	0.1192	0.1192	0.1192
2	RF vs. R3	0.1205	0.7230	0.6025	0.3615	0.3615
3	RF 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

In this study, we introduced a novel method (Rule 1) by combining the random subspace section with random coverage to facilitate diversity in rules using the covering algorithm for inducing unordered rule sets. We compared this method with the random forest algorithm and its counterpart for unordered rule induction (Rule 2) and an algorithm for combining all three diversity-facilitating (Rule 3) methods in one algorithm: bagging, random subset selection, and random coverage. Although the results 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 (Rule 1) warrants further attention. One venue forward would be to set up an experiment with more datasets to be able to answer the question of whether differences exist between the methods or whether there are random fluctuations. It would also be interesting to investigate whether one can devise a better method than just randomly selecting examples to keep in the random coverage method. One idea could be to keep examples that belong to the minority class of the rule and hence face the danger of (erroneously) being classified as belonging to the majority class. Another potential tack would be to introduce weights to examples, which 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)*, 2000, pp. 1-15.

[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 masters 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.