Prediction rule ensembles in R

M. Fokkema[1]
[1]Universiteit Leiden

Abstract

Prediction rule ensembles are a highly interpretable method for regression and classification. Prediction rule ensembles consist of a small set of simple decision rules, which are easy to evaluate and apply in practice. The current paper discusses rule ensemble methodology, and introduces the unbiased rule prediction ensemble (upre) algorithm. The upre algorithm and its software implementation are largely based on the prediction rule ensemble methodology of J. H. Friedman and Popescu (2008), with some added features and improvement. To illustrate the algorithm and the `R` package `upre`, upre will be applied to real datasets and the results will presented. The software is well documented and freely available. It is hoped that readers, after reproducing the examples, will wave their hands in the air like they just don't care and exclaim 'Upre? I think it's super!'.

Introduction

Prediction rule ensembles provide accurate and interpretable methods for regression and classification. Prediction rules are logical statements of the form if [conditions] then [prediction], which are highly usable in clinical decision making. The rules in prediction rule ensembles can be seen as very simple decision trees (Fokkema, Smits, Kelderman, & Penninx, 2015; Katsikopoulos, Pachur, Machery, & Wallin, 2008). Decision trees can be derived by, for example, the classification and regression trees (CART; Breiman, Friedman, Olshen, & Stone, 1984) algorithm, and such trees are well known for their interpretability and applicability in practical in decision making. However, single trees are also known for their instability: small changes in the training data may cause large changes in the resulting tree (e.g., Dannegger, 2000).

A powerful solution to this problem of instability is combining the predictions of many single trees, with techniques like bagging, boosting and random forests. Ensembling trees has been shown to substantially improve predictive accuracy (e.g., Breiman, 1996b; T. Dietterich, 2000; Strobl, Malley, & Tutz, 2009). However, the tree ensembles generally consist of a large number of trees, and are therefore difficult to interpret, let alone apply without

the help of statistical software. A trade-off between accuracy and interpretability seems to apply: single trees provide high interpretability but lower accuracy, whereas ensembles provide high accuracy but lower interpretability.

Rule-based ensembles aim to balance accuracy and interpretability: accuracy is improved by ensembling predictions and interpretability is improved by using prediction rules instead of full trees. Several algorithms for deriving rule-based ensembles have been developed. Most are aimed exclusively at classification, like ROCCER, (Prati & Flach, 2005), SLIPPER (Cohen & Singer, 1999), Lightweight Rule Induction (Weiss & Indurkhya, 2000) and C4.5rules (Quinlan, 2014). In contrast, the Rulefit (J. H. Friedman & Popescu, 2008), ENDER (Dembczyński, Kotłowski, & Słowiński, 2010) and Node Harvest (Meinshausen, 2010) algorithms can be applied to both classification and regression problems. Among these three, Rulefit is most explicitly aimed at balancing accuracy and interpretability, as it employs regularized regression to select only a small set of prediction rules for the final ensemble. In contrast, ENDER and Node Harvest produce ensembles with a large number of rules, limiting interpretability of the ensemble.

The aim of the current paper is to introduce the unbiased prediction rule ensemble (upre) algorithm and its software implementation. The upre algorithm is largely based on Rulefit, as described by (J. H. Friedman & Popescu, 2008), and derives a prediction rule ensembles in two stages. In the first stage, a large initial ensemble of rules is generated by taking the nodes from trees that are grown on a large number of samples drawn from the training dataset. This will be further described in the subsection Rule generation. In the second stage, a small final ensemble of prediction rules is selected by means of penalized regression, which will be further described in the subsection Rule selection. In the subsection Comparison with Rulefit, the differences and similarities between upre and Rulefit will be discussed. In the subsection Illustration, the upre algorithm will be applied to several real datasets, to illustrate the algorithm and its `R` implementation. In the Simulation section, the predictive accuracy of upre will be assessed and compared with that of Rulefit and Node Harvest, using simulated datasets. In the Discussion, results will be summarized, and further research and future developments will be discussed.

Rule generation

The upre algorithm derives prediction rules from an ensemble of trees, which are sequentially grown on random samples of the training data. The trees grown by upre are conditional inference trees or ctrees (Hothorn, Hornik, & Zeileis, 2006; ?, ?). Although tree-based ensemble algorithms generally use CART for growing trees, CART uses biased variable selection criteria: given two equally informative input variables, the variable with a larger number of unique and/or missing values has a higher probability of being selected as a splitting variable (e.g., Kim & Loh, 2001; Strobl, Boulesteix, Kneib, Augustin, & Zeileis,

2008). The ctree algorithm provides unbiased variable selection, by using statistical tests for parameter instability to select splitting variables. For further details on the parameter instability tests and split selection, readers are referred to ?, ?.

An example ctree is presented in Figure . The first node of a tree does not represent an informative rule, as it applies to all observations in the sample. Therefore, although the tree in Figure consists of nine nodes, the following eight prediction rules are derived from it:

$$r_1(\mathbf{x}) = I(x_4 \leq 82)$$
$$r_2(\mathbf{x}) = I(x_4 \leq 82 \cdot x_3 \in \{a, c\})$$
$$r_3(\mathbf{x}) = I(x_4 \leq 82 \cdot x_3 \in \{b, d, e\})$$
$$r_4(\mathbf{x}) = I(x_4 \leq 82 \cdot x_3 \in \{b, d, e\} \cdot x_4 \leq 77)$$
$$r_5(\mathbf{x}) = I(x_4 \leq 82 \cdot x_3 \in \{b, d, e\} \cdot x_4 > 77)$$
$$r_6(\mathbf{x}) = I(x_4 > 82)$$
$$r_7(\mathbf{x}) = I(x_4 > 82 \cdot x_5 \leq \text{seldom})$$
$$r_8(\mathbf{x}) = I(x_4 > 82 \cdot x_5 > \text{seldom})$$

Where $\mathbf{x}$ is random vector of n input variables, I is an indicator of the truth of its argument, and $r_m(\mathbf{x})$ denotes prediction rule m, taking a value of 0 when its conditions do not apply, and a value of 1 when its conditions apply. Note that the tree in Figure shows that input variables used for growing trees may be continuous (like x_4), unordered categorical (like x_3) or ordered categorical (like x_5). Also, that the same input variable may appear multiple times in the same rule (like x_4 in $r_4(\mathbf{x}) or r_5(\mathbf{x})$).

To derive the ensemble of trees, approaches similar to bagging (Breiman, 1996a), boosting (Freund & Schapire, 1995), random forests (Breiman, 2001), or a combination can be used. In either appraoch, a user-specified number of random samples (500, by default) of size η are drawn from the training data with or without replacement. Setting $\eta = 1$ will result in sampling with replacement, or bootstrap sampling. Setting $\eta < 1$ will result in sampling without replacement, or subsampling. Trees are grown on the random samples in a sequential fashion, with learning rate $\nu = 0, \ldots, 1$. The learning rate ν corresponds to the shrinkage parameter in boosting, and controls the influence of earlier trees in the induction of new trees. Let $m = 1, \ldots, M$ denote the random sample of observations drawn from the training data, and $f_m(\mathbf{x})$ the predictive function from the tree grown on subsample m. Then every tree $m > 1$ is grown using outcome variable u_m, instead of the original outcome variable y:

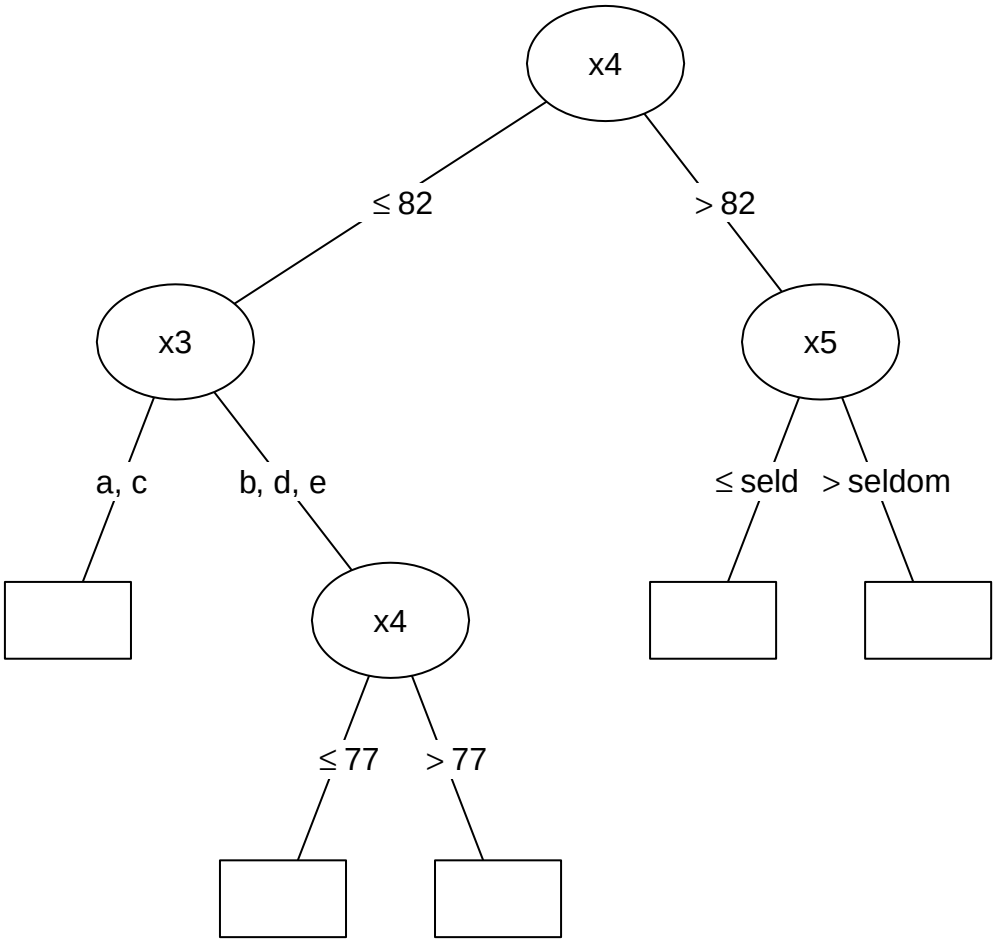

Figure 1. Example conditional inference tree. Note that a conditional inference tree would normally depict the distribution of the outcome variable y in each of the terminal nodes. For generating the initial ensemble of prediction rules, only the tree structure, not the distribution of the outcome variable is used, which is therefore omitted here.

$$u_m = y - \sum_{k=1}^{m-1} \nu \cdot f_k(\mathbf{x}) \tag{1}$$

Note that setting $\nu = 0$ removes the influence of earlier trees, with each tree being

grown using the original values of the outcome variable y. Setting $\nu = 1$ maximizes the influence of earlier trees. J. Friedman and Popescu (2003) have found small, nonzero values of ν (e.g., .01) to perform best for ensembles of small decision trees. In addition, they found their results to be fairly insensitive to the value selected for η, provided it was small (i.e., $\eta < N/2$ for $N \leq 500$ and $\eta < 10 \cdot \sqrt{N}$ for $N > 500$). The prediction rules in a upre ensemble can be based on a bagged ensemble by setting $\nu = 0$ and $\eta = 1$. The rules can be based on a random forest by setting $\nu = 0$, $\eta = 1$ and the number of input variables to be selected as candidates for each split to values $< n$. Note that ν, η and the number of input variables to be selected as candidates for each split can be varied independently from each other, to create a hybrid of bagging, boosting, and random forests.

Note that the original bagging and random forest algorithms make use of unpruned CART trees, which can be mimicked in the upre software implementation by setting the maximum tree depth to infinity. However, in situations with a large number of observations and few variables, Lin and Jeon (2006) have found predictive accuracy of random forests to improve when tree size is limited. In addition, smaller trees provide shorter prediction rules, which are easier to interpret. Therefore, by default upre restricts maximal tree depth to 3, thereby limiting the maximum number of conditions of rules to 3. In effect, this limits the interactions that can be captured by the rule ensemble to second-order and three-variable interactions. Users may provide any other integer value for maximum tree depth, as well.

After the ensemble of trees is generated, every node from every tree is included in the initial rule ensemble, after removing perfectly collinear rules (i.e., rules with conditions or support identical to that of earlier rules).

In addition, users may choose the original input variables to be added as linear terms to the initial ensemble. Although non-linear functions of input variables are relatively easy for rules to approximate, linear functions are not. Therefore, addition of the input variables may improve sparsity and predictive accuracy of the final ensemble. To reduce the effect of possible outliers, input variables may be winsorized before entering the predictive model. The linear terms are given by:

$$l_j(x_j) = \min(\delta_j^+, max(\delta_j^-, x_j)) \tag{2}$$

where j is an indicator for the input variable, and δ_j and δ_j are the β and $(1 - \beta)$ quantiles of the distribution of x_j in the training data. The value chosen for β reflect user's prior suspicion of the fraction of outliers. By default β is set to .025. Setting $\beta = 0$ will result in the input variables not being winsorized.

Selection of the final ensemble

The initial ensemble consists of a large number of rules and/or linear functions, only a few of which may contribute to predictive accuracy on future observations. Therefore, coefficients for the rules and/or linear terms for the final predictive model are derived using penalized regression. When both rules and linear terms are included, the predictive model becomes:

$$F(\mathbf{x}) = \hat{a}_0 + \sum_{k=1}^{K} \hat{a}_k r_k(\mathbf{x}) + \sum_{j=1}^{n} \hat{b}_j l_j(x_j) \tag{3}$$

Where $1 \dots j \dots n$ is an index for the input variable and $1 \dots k \dots K$ is an index for the rule in the initial ensemble. The coefficient values are estimated by means of penalized regression. When the final model is selected by lasso regression (Tibshirani, 1996), which is the default in the upre algorithm, the estimates satisfy the following minimization criterion:

$$(\{\hat{a}_k\}_0^K, \{\hat{b}_j\}_1^n) = \arg \min_{\{\hat{a}_k\}_0^K, \{\hat{b}_j\}_1^n} \sum_{i=1}^{N} L\left(y_i, \hat{a}_0 + \sum_{k=1}^{K} \hat{a}_k r_k(\mathbf{x}_i) + \sum_{j=1}^{n} \hat{b}_j l_j(x_i j)\right) + \lambda \left(\sum_{k=1}^{K} |\hat{a}_k| + \sum_{j=1}^{n} |\hat{b}_j|\right) \tag{4}$$

The first term is the prediction risk over the N training observations, and the second term is a penalty for non-zero coefficients of rules and linear terms. The influence of the penalty term is determined by the value of $\lambda \geq 0$. The value of λ is taken to be that which minimizes the prediction risk on a separate validation dataset, or by multi-fold cross validation on the training data. The penalty term in 5 forces many coefficients to be set to zero, providing a final ensemble consisting of a small set of rules and/or linear terms. Users can also select ridge or elastic net regression (Zou & Hastie, 2005) to derive the final ensemble. The penalty term in Equation 5 would then replaced by a ridge or elastic net counterpart, respectively penalizing squared values of $\hat{a}_k$ and $\hat{b}_j$ instead, or as well as, absolute values.

The penalty term in 5 is affected by linear transformations of the input variables: generally, terms with smaller variance are more heavily penalized. Users can give linear terms the same influence on the penalty term as a typical rule, by selecting their normalized versions to be used in the estimation (Equation 5):

$$l_j(x_j) < -0.4 * l_j(x_j)/std(l_j(x_j)) \tag{5}$$

Interpreting the final ensemble: variable importances and interactions

Importances. The estimated coefficients for rules and/or linear terms in the final ensemble provide an unstandardized global measure of their contribution to the predictive

model. Importances are provide standardized measures of their contribution to the predictive model, and can be calculated for rules, linear terms, as well as input variables (as input variables may appear in the final model as part of one or more rules, as well as a linear term). Variable importances provide a measure of the extent to which individual input variables influence the predictions of the final rule ensemble. Importances can be calculated globally (averaged over the whole training dataset) or locally (over a selected subregion of the input variable space). The global importance of a rule r_k is given by

$$I_k = |\hat{a}_k| \cdot \sqrt{s_k(1 - s_k)} \tag{6}$$

where s_k is the support of rule r_k on the training data, given by

$$s_k = \frac{1}{N} \sum_{i=1}^{N} r_k(x_i) \tag{7}$$

The global importance of linear term $l_j(x_j)$ is given by

$$I_j = |\hat{b}_j| \cdot std(l_j(x_j)) \tag{8}$$

where $std(l_j(x_j))$ is the standard deviation of the linear term $l_j(x_j)$ (Equation 3) in the training data. The importance of an input variable J_j is given by the sum of the importance of its linear term I_j, and a weighted sum of the importances of the rules I_k in which the variable appears:

$$J_j = I_j + \sum_{x_j \in r_k} I_k / m_k \tag{9}$$

where m_k is the number of variables that define rule k. The second term in Equation 9 shows that the importance of a rule is distributed equally over the input variables appearing in the rule[1].

Local importances for a selected subregion S of the input variable space can be calculated by replacing the global support $_k$ in Equation **??** by the support in the selected subregion, which is given by

$$s_k(S) = \frac{1}{|S|} \sum_{x_i \in S} r_k(x_i) \tag{10}$$

where $|S|$ is the cardinality (number of training set observations) in subregion S. Similarly, the local importance of a linear term over a selected subregion can be calculated by

[1]Note that when x_j appears multiple times in the conditions of r_k, Equation 9 is not completely accurate. In those instances, the importance of r_k that is distributed to the importance J_j equals $\frac{\text{the number of times } x_j \text{appears in the conditions of} r_k \times I_k}{\text{the total number of conditions in} r_k}$

replacing the global standard deviation $std(l_j(x_j))$ in Equation **??** by its standard deviation in the selected subregion:

$$std(l_j(x_j), S) = \sqrt{\frac{1}{|S| - 1} \sum_{x_i \in S} (l_j(x_i j) - \bar{l}_j(x_j, S))^2} \tag{11}$$

where $\bar{l}_j(x_j, S)$ is the mean of $l(x_j)$ in subregion S.

Local importances can also be calculated for a single point $\mathbf{x}$. The local importance of rule r_k at a single point $\mathbf{x}$ is given by

$$I_k(\mathbf{x}) = |\hat{a_k}| \cdot |r_k(\mathbf{x}) - s_k|) \tag{12}$$

The local importance of linear term $l_j(x_j)$ at a single value x_j is given by

$$I_j(x_j) = |\hat{b_j}| \cdot |l_j(x_j) - \bar{l}_j|) \tag{13}$$

where $\bar{l}_j$ is the mean of $l_j(x_j)$ over the training data. The quantities in Equations **??**, **??**, **??** and **??** represent the absolute change in the (average) prediction $F(\mathbf{x})$ when the corresponding rule or linear term is dropped from the predictive model (Equation 4), when the intercept would be adjusted accordingly (i.e., $\hat{a}_0 < -\hat{a}_0 + \hat{a}_k s_k$ for rules and $\hat{a}_0 < -\hat{a}_0 + \hat{b}_j \bar{l}_j$ for linear terms). The local importance of input variable x_j at each point $\mathbf{x}$ is given by

$$J_j(\mathbf{x}) = I_j(\mathbf{x}) + \sum x_j \in r_k I_k(\mathbf{x})/m_k \tag{14}$$

Partial dependence plots

The influence of individual predictor variables on the predictions of the final rule ensemble can also be represented visually in partial dependence plots

Interaction effects. One of the main advantages of tree-based methods is the automatic detection of non-linear and/or interaction effects. However, the presence of interactions or non-linear effects of single variables cannot simply be inferred from inspection of the rules in the final ensemble. On the one hand, variables involved in interaction effects do not necessarily appear in the same rules, and on the other hand, variables appearing in the same rules do not necessarily produce an interaction effect on the outcome variable. To assess non-linear effects and multi-way interactions of predictor variables, partial dependence plots and statistical tests are provided.

In addition, the presence of interactions of (user-specified subsets of) predictor variables can be statistically tested. For these tests, a null model involving main effects of predictor variables only, is compared with a model involving interactions.

To disencourage spurious interactions entering the model, F&P (section 8.2) use a strategy to put an incentive on variables entering a tree, that do not appear earlier in the tree. Meinshausen uses a random forest methodology for deriving the trees for the initial ensemble, which may have the same results: by selecting only a subset of the predictor variables for growing each tree, interaction effects will be disencouraged.

Comparison with Rulefit

- ctree instead of CART trees - Rulefit uses an approach that can be best described as a combination of bagging and boosting: it takes subsamples[2] of size η of the training data, and grows trees in a sequential fashion with learning rate $\nu = 0, \ldots, 1$. - Rulefit allows for inducing trees in a manner that can be best described as a hybrid of bagging (Breiman, 1996a) and boosting (J. H. Friedman, 2001). In some situations, bagged ensembles of trees may be outperformed by random forests in terms of predictive accuracy (T. G. Dietterich, 2000). Also, Node Harvest derives prediction rules from the nodes of random forest trees. Therefore, upre allows for inducing trees in a bagging, boosting or random forest type manner, or a hybrid of the three. - Rulefit also provides forward stagewise regression - Current implementation of rulefit does not provide user-friendly interface for rules inspection, does not provide value of lambda or intercept, and does not allow user to select a different values for lambda (e.g., lamba min or lambda 1se).

Miscellaneous

RuleFit returns the cross-validated model selection criterion value with standard error (when test.reps ¿ 1), and number of terms in the resulting model, in the command line. The penalized regression is performed on the whole set of training observations. However, the value of the penalty parameter(s) is chosen to be the value that minimizes future prediction risk, estimated in a separate sample not used in training, or by full multi-fold cross validation.

"3.2. Missing values. An interesting property of NH is its natural ability to cope with missing values. Once a fit is obtained, predictions for new data can be obtained without use of imputation techniques or surrogate splits. To fit the node harvest estimator with missing data, we replace missing values in the matrix X by the imputation technique described in ...."

Users are advices to not use the lambda path determined by `glmnet`, include a way to determine own path?

[2]With subsampling, observations are drawn without replacement from the full dataset, so strictly speaking, Rulefit does not use a bagging approach, as this would involve bootstrap samples of the dataset.

Extensions to random-effect models

Schelldorder, Meier & Buhlmann (2014) propose a two-step approach for high-dimensional GLMMs: In the first step, variable screening is performed, selecting a set of candidate active variables. In the second step, refitting by maximum likelihood (ML) estimation is performed to get accurate parameter estimates. Groll & Tutz (2010) propose a method that works by combining gradient ascent optimization with the Fisher scoring algorithm and is based on the approach of Goeman (2010).

Alternative rule ensemble methods

Meinshausen's Node harvest algorithm derives an initial ensemble Q which is very large (thousands of rules). The rules in the initial ensemble can be generated at random from the training data, without using a response variable. However, Meinshausen (2010) suggests to derive the rules from the trees in a random forest (Breiman, 2001). However, the trees are fitted on subsamples of the data (of size n/10), rather than on bootstrap samples. Next, the nodes that satisfy maximal interaction order and minimal node size constraints are added to the initial ensemble, provided that they are not already present in the set. If two or more nodes in the initial ensemble contain the exact same set of training observations, only one is randomly selected and included in the ensemble.

Node Harvest uses only the nodes (rules) in which a new observation 'falls' for prediction, requiring only a small number of rules for a single observation. However, future observations may 'fall' into any of the nodes of the initial ensemble, so all rules from the initial (and large) ensemble have to be retained to allow for prediction of future observations.

ROCCER (Prati & Flach, 2005), SLIPPER (Cohen & Singer, 1999), LRI (Weiss & Indurkhya, 2000), and C4.5rules (Quinlan, 2014) are for classification problems only. (Dembczyński et al., 2010) can be used for both classification and regression. None of these methods are implemented in statistical software like R of SPSS.

Lightweight Rule Induction (Weiss & Indurkhya, 2000): "A light weight rule induction method is described that generates compact Disjunctive Normal Form (DNF) rules. Each class has an equal number of unweighted rules. A new example is classified by applying all rules and assigning the example to the class with the most satisfied rules. The induction method attempts to minimize the training error with no pruning. An overall design is specified by setting limits on the size and number of rules. During training, cases are adaptively weighted using a simple cumulative error method. The induction method is nearly linear in time relative to an increase in the number of induced rules or the number of cases. Experimental results on large bench-mark datasets demonstrate that predictive performance can rival the best reported results in the literature."

SLIPPER (Cohen & Singer, 1999): "We describe SLIPPER - a new rule learner that

generates rulesets by repeatedly boosting a simple, greedy rule builder. Like the rulesets built by other rule learners, the ensemble of rules created by SLIPPER is compact and comprehensible. This is made possible by imposing appropriate constraints on the rule-builder, and by use of a recently proposed generalization of Ad-aboost called confidence-rated boosting. In spite of its relative simplicity, SLIPPER is highly scalable, and an effective learner. Experimentally, SLIPPER scales no worse than O(n log n), where n is the number of examples, and on a set of 32 benchmark problems, SLIPPER achieves lower error rates than RIPPER 20 times, and lower error rates than C4.5rules 22 times."

(Quinlan, 2014) C4.5rules is for classification only

SLIPPER and ENDER take a boosting approach to rule induction.

ENDER (Dembczyński et al., 2010): "We believe that ENDER can still be used for interpretation purposes. One way is to follow the approach given in Friedman and Popescu (2008) that relies on a postprocessing phase in which the rules are refitted by using the lasso regularization."

References

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (1996b). Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, *24*(6), 2350–2383.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. New York: Wadsworth.

Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. In *Proceedings of the national conference on artificial intelligence* (pp. 335–342).

Dannegger, F. (2000). Tree stability diagnostics and some remedies for instability. *Statistics in medicine*, *19*(4), 475–491.

Dembczyński, K., Kotłowski, W., & Słowiński, R. (2010). ENDER: A statistical framework for boosting decision rules. *Data Mining and Knowledge Discovery*, *21*(1), 52–90.

Dietterich, T. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems*, *1857*, 1–15.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, *40*(2), 139–157.

Fokkema, M., Smits, N., Kelderman, H., & Penninx, B. W. (2015). Connecting clinical and actuarial prediction with rule-based methods. *Psychological assessment*, *27*(2), 636.

Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37).

Friedman, J., & Popescu, B. (2003). *Importance sampled learning ensembles* [Technical Report]. Stanford University. (http://www-stat.stanford.edu/ jhf/ftp/isle.pdf)

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of*

Statistics, 1189–1232.

Friedman, J. H., & Popescu, B. E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 916–954.

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, *15*(3), 651–674.

Katsikopoulos, K. V., Pachur, T., Machery, E., & Wallin, A. (2008). From meehl to fast and frugal heuristics (and back) new insights into how to bridge the clinicalactuarial divide. *Theory & Psychology*, *18*(4), 443–464.

Kim, H., & Loh, W.-Y. (2001). Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, *96*(454), 589–604.

Lin, Y., & Jeon, Y. (2006). Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, *101*(474), 578–590.

Meinshausen, N. (2010). Node harvest. *The Annals of Applied Statistics*, 2049–2072.

Prati, R. C., & Flach, P. A. (2005). Roccer: An algorithm for rule learning based on roc analysis. In *Ijcai* (pp. 823–828).

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. *BMC bioinformatics*, *9*(1), 1.

Strobl, C., Malley, J., & Tutz, G. (2009). An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, *14*(4), 323.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–2f88.

Weiss, S. M., & Indurkhya, N. (2000). Lightweight rule induction. In *Proceedings of the seventeenth international conference on machine learning* (pp. 1135–1142).

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.