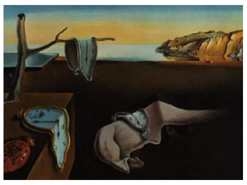

Journal of Statistical Software
February 2020, Volume 92, Issue 12. doi: 10.18637/jss.v092.i12

Fitting Prediction Rule Ensembles with R Package pre

Marjolein Fokkema
Universiteit Leiden

Abstract

Prediction rule ensembles (PREs) are sparse collections of rules, offering highly interpretable regression and classification models. This paper shows how they can be fitted using function `pre` from R package **pre**, which derives PREs largely through the methodology of Friedman and Popescu (2008). The implementation and functionality of `pre` is described and illustrated through application on a dataset on the prediction of depression. Furthermore, accuracy and sparsity of `pre` is compared with that of single trees, random forests, lasso regression and the original RuleFit implementation of Friedman and Popescu (2008) in four benchmark datasets. Results indicate that `pre` derives ensembles with predictive accuracy similar to that of random forests, while using a smaller number of variables for prediction. Furthermore, `pre` provided better accuracy and sparsity than the original RuleFit implementation.

Keywords: prediction rules, ensemble learning, decision trees, lasso penalty, R.

1. 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 easy to use in decision making. The prediction rules can be depicted as very simple decision trees, further improving interpretability (e.g., Fokkema, Smits, Kelderman, and Penninx 2015).

Several algorithms for deriving decision trees are available, an early example being the classification and regression tree algorithm (CART; Breiman, Friedman, Olshen, and Stone 1984). Although CART trees are easy to interpret, they suffer from two disadvantages: biased variable selection and instability. Although the variable selection bias has been addressed by several later tree induction algorithms, like for example the conditional inference trees algorithm of Hothorn, Hornik, and Zeileis (2006), the problem of instability is shared by all tree induction algorithms. Instability here means that small changes in the training data may yield major changes in the resulting tree.

A powerful solution to the instability problem is combining the predictions of single trees through ensembling, which has been found to substantially improve predictive accuracy (e.g., Breiman 1996; Dietterich 2000; Strobl, Malley, and Tutz 2009). However, the resulting ensembles generally consist of a large number of trees and are therefore difficult to interpret and apply. A trade-off between accuracy and interpretability seems to apply: single trees provide better interpretability, whereas tree ensembles provide better accuracy.

Prediction rule ensembles (PREs) aim to optimize accuracy as well as interpretability, by creating ensembles with a small number of simple trees or rules. Several algorithms for deriving PREs have been developed, most exclusively aimed at classification, like SLIPPER (Cohen and Singer 1999) and lightweight rule induction (Weiss and Indurkhya 2000). Alternatively, the RuleFit (Friedman and Popescu 2008), ENDER (Dembczyński, Kotłowski, and Słowiński 2010) and node harvest (Meinshausen 2010) algorithms can be applied to classification as well as regression problems. The RuleFit algorithm generates a large initial ensemble of rules from a boosted tree ensemble and selects a sparse final rule ensemble using lasso regression. This approach yields ensembles that are competitive in accuracy with, for example, boosted tree ensembles and random forests (Friedman and Popescu 2008; Joly, Schnitzler, Geurts, and Wehenkel 2012; Shimokawa, Li, Yan, Kitamura, and Goto 2014; Yang, Zhang, Chen, Chen, Li, and Lu 2008).

The aim of the current paper is to introduce function `pre` from R (R Core Team 2019) package **pre** (Fokkema and Christoffersen 2020), which provides a completely R-based implementation of the algorithm of Friedman and Popescu (2008). Package **pre** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=pre`. Although the **RuleFit** program (Friedman and Popescu 2012) already provides a fast implementation of the algorithm, `pre` provides a number of potential advantages: First, in addition to the CART algorithm, it allows for employing the unbiased recursive partitioning algorithms of Hothorn *et al.* (2006) and Zeileis, Hothorn, and Hornik (2008) to generate rules. Second, in addition to continuous and binary outcomes, `pre` allows for the analysis of count, multinomial, multivariate continuous and survival outcomes. Third, in addition to bagging and boosting, `pre` allows for generating prediction rules through a random-forest style approach. Fourth, `pre` is completely R-based, providing R users with a more familiar interface and more easily accessible results and documentation. The first and last advantages, however, come at a computational cost, yielding longer computation times for `pre` than for the original **RuleFit** program.

In what follows, the implementation (Section 2) and functionality (Section 3) of `pre` will be described. In Section 4, application of `pre` will be illustrated using an existing dataset on the prediction of depressive symptomatology. Also, several examples will illustrate how well-known tree ensemble approaches can be mimicked in rule generation. In Section 5, the performance of `pre` will be compared with that of single trees, random forests, lasso penalized linear regression and the original RuleFit implementation in four benchmark datasets. In Section 6, the properties of `pre` and other software packages for deriving PREs will be discussed and compared.

2. Implementation

2.1. Rule generation

Following the methodology of Friedman and Popescu (2008), `pre` first generates a large ensemble of decision trees, from which an initial ensemble of prediction rules is derived. To induce trees, `pre` by default employs function `ctree` from the **partykit** package (Hothorn and Zeileis 2015). Alternatively, function `glmtree` (also from package **partykit**; Zeileis *et al.* 2008) or function `rpart` from package **rpart** (Therneau, Atkinson, and Ripley 2019) can be employed. Function `rpart` implements the original CART algorithm of Breiman *et al.* (1984), whereas functions `ctree` and `glmtree` implement unbiased recursive partitioning procedures, which address the variable selection bias mentioned earlier, through separating variable and cut-point selection. Function `glmtree` fits GLMs with different parameter estimates in every node. To obtain a tree with constant fits in the nodes with `glmtree`, an intercept-only GLM is specified.

Functions `ctree`, `glmtree` and `rpart` employ different criteria for variable and/or cut-point selection and will yield somewhat different tree structures, given the same data. Although `glmtree` will yield the longest computation times, it may yield slightly higher accuracy. For further details on variable and cut-point selection criteria employed by the different algorithms, the reader is referred to Hothorn *et al.* (2006), Zeileis *et al.* (2008) and Therneau *et al.* (2019).

To illustrate rule generation, Figure 1 depicts an example tree. From this tree, the following set of rules can be derived:

$$\begin{aligned}
r_1(\mathbf{x}) &= I(x_4 \leq 82.7),\\
r_2(\mathbf{x}) &= I((x_4 \leq 82.7) \cdot (x_3 \in \{a, c\})),\\
r_3(\mathbf{x}) &= I((x_4 \leq 82.7 \cdot (x_3 \in \{b, d, e\})),\\
r_4(\mathbf{x}) &= I(x_4 > 82.7),\\
r_5(\mathbf{x}) &= I((x_4 > 82.7) \cdot (x_5 \leq \text{seldom})),\\
r_6(\mathbf{x}) &= I((x_4 > 82.7) \cdot (x_5 > \text{seldom})),
\end{aligned}$$

where $r_k(\mathbf{x})$ denotes prediction rule k ($k = 1, \dots, K$), taking a value of 1 if its conditions apply, and a value of 0 if not; $\mathbf{x}$ denotes a random vector of p input variables; x_j denotes input variable j ($j = 1, \dots, p$); and I is a function denoting the truth of its argument. Note that, with exception of the root node, a rule is generated for every node in the tree, not just for the terminal nodes. This also yields redundant rules: for example, the first and fourth rule above are perfectly collinear, that is $r_4(\mathbf{x}) = 1 - r_1(\mathbf{x})$. Therefore r_4 will be omitted from the initial ensemble. Similarly, rules that are identical to earlier generated rules are also removed from the initial ensemble. Furthermore, as shown in Figure 1, input variables may be continuous (like x_4), unordered categorical (like x_3) or ordered categorical (like x_5).

The ensemble of decision trees can be generated in an approach similar to bagged, boosted or random forest tree ensembles, or a combination thereof. By default, `pre` draws $B = 500$ random sub-samples from the training data and grows a tree on each sample. As in bagging, `pre` allows for employing bootstrap sampling (i.e., sampling with replacement), but subsampling has been found to yield lower inclusion frequencies for noise variables (De Bin,

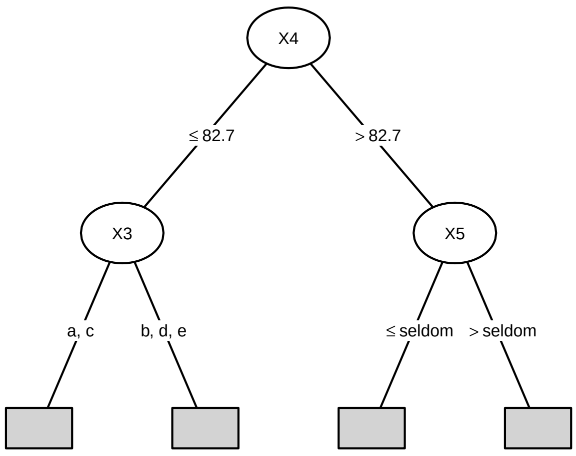

Figure 1: Example tree. Information on the distribution of the outcome variable in the terminal nodes is omitted, as only the tree structure is used for generating rules.

Janitza, Sauerbrei, and Boulesteix 2016). In addition to random sampling of observations, `pre` allows for random-forest style sampling of predictor variables for split selection through specification of an `mtry` argument. By default, however, all predictor variables are considered for split selection.

To apply boosting, a learning rate (or shrinkage parameter) ν can be specified, which controls the influence of earlier trees on the induction of new trees. If $\nu > 0$, a gradient boosting approach is employed, where the tree in iteration b ($b = 1, \ldots, B$) is grown on the pseudo response $\tilde{y}_b$, instead of the original response y.

For a continuous response variable y, the pseudo response in iteration b is given by:

$$\tilde{y}_b = y - \sum_{m=1}^{b-1} \nu \cdot f_m(\mathbf{x}), \tag{1}$$

where $f_m(\mathbf{x})$ are the predictions from the regression tree fitted in iteration m ($m = 1, \ldots, b-1$). For a multivariate continuous response variable $\mathbf{y}$, Equation 1 would also be employed, but with a multivariate pseudo response $\tilde{\mathbf{y}}_b$ and multivariate predictions $f_m(\mathbf{x})$. For a binary (0−1 coded) response variable, regression trees are fitted to a continuous pseudo response, which is given by:

$$\tilde{y}_b = \frac{y - p_{b-1}}{\sqrt{p_{b-1}(1 - p_{b-1})}}, \tag{2}$$

where

$$p_{b-1} = \frac{1}{1 + e^{-\eta_{b-1}}}. \tag{3}$$

In the first iteration, the value of η is given by:

$$\eta_0 = \log\left(\frac{\bar{p}}{1 - \bar{p}}\right), \tag{4}$$

where $\bar{p}$ is the (possibly weighted) mean of y. In subsequent iterations ($b > 1$), η is given by:

$$\eta_{b-1} = \eta_0 + \sum_{m=1}^{b-1} \nu \cdot f_m(\mathbf{x}). \tag{5}$$

For multinomial response variables, gradient boosting is applied by coding the response as a set of $0-1$ coded indicator variables, one for each response category, and applying Equations 2 through 5 to obtain a multivariate pseudo response.

For count response variables, regression trees are also employed and the pseudo response is given by:

$$\tilde{y}_b = y - \lambda_{b-1}, \tag{6}$$

where

$$\lambda_{b-1} = e^{\eta_{b-1}}. \tag{7}$$

In the first iteration, the value of η is given by:

$$\eta_0 = \log\left(\bar{\lambda}\right), \tag{8}$$

where $\bar{\lambda}$ is the (possibly weighted) mean of y. In subsequent iterations ($b > 1$), η is given by Equation 5.

For the gradient boosting approach described above, functions `ctree` or `rpart` are employed. Alternatively, function `glmtree` can be employed, allowing for application of the learning rate through including an offset in the GLM (i.e., a predictor with a fixed coefficient of 1). That is, in every iteration, a GLM-based recursive partition is fit on the response y with the offset in each iteration b given by:

$$\eta_{b-1} = \sum_{m=1}^{b-1} \nu \cdot f_m(\mathbf{x}), \tag{9}$$

where $f_m(\mathbf{x})$ are the predictions on the scale of the linear predictor from the tree fitted in iteration m. Whereas gradient boosting with `ctree` or `rpart` yields shorter computation times, the use of `glmtree` may yield a somewhat more accurate final ensemble.

Based on the findings of Friedman (2001) and Friedman and Popescu (2003), who found small non-zero values of the learning rate to perform best for ensembles of small decision trees, the learning rate ν of `pre` is set to 0.01, by default.

Although the original bagging and random forest algorithms made use of unpruned trees, limiting tree size has been found to yield better predictive accuracy (e.g., Lin and Jeon 2006). Smaller trees also yield shorter prediction rules, which are easier to interpret. Function `pre` therefore generates trees with a maximum depth of three by default, yielding a maximum of three conditions per rule and restricting interactions that can be captured to first- and second-order ones. Other values for maximum tree depth can be specified by the user.

2.2. Selection of the final ensemble

After the ensemble of decision trees is generated, every node from every tree is included as a rule in the initial rule ensemble. Rules that are perfectly collinear with earlier rules are omitted from the initial ensemble, by default. Furthermore, all predictor variables are included as linear terms in the initial ensemble, by default. This may improve sparsity and/or accuracy of the final ensemble, as rules may have difficulty approximating purely linear functions of input variables. Alternatively, the ensemble may be specified to include either rules or linear terms only. To reduce the effect of possible outliers, continuous and ordered categorical predictor variables are winsorized before inclusion as linear terms in the initial ensemble:

$$l_j(x_j) = \min(\delta_j^+, \max(\delta_j^-, x_j)), \tag{10}$$

where δ_j^- and δ_j^+ represent the β and $(1-\beta)$ quantiles of the distribution of predictor variable x_j in the training data. By default, β is set to 0.025, but other values may be specified by the user.

Unordered factors are included in the initial ensemble as $(q_j - 1)$ $0-1$ coded variables, where q_j corresponds to the number of levels of predictor variable j. The resulting initial ensemble consists of a large number of base learners (rules and/or linear terms), of which only a small subset may actually contribute to predictive accuracy. Therefore, coefficients for the base learners are estimated using penalized regression. By default, `pre` employs the lasso penalty, but ridge or elastic net penalties can also be selected by the user.

As the lasso penalty more heavily penalizes predictors with smaller variance, linear terms are normalized before estimation, by default:

$$l_j(x_j) \leftarrow 0.4 \cdot \frac{l_j(x_j)}{sd(l_j(x_j))}, \tag{11}$$

where $sd(l_j(x_j))$ is the sample standard deviation of the linear term (Equation 10) and 0.4 represents the standard deviation of a typical rule.[1]

If both rules and linear terms are included in the ensemble, the final predictive model is given by:

$$F(\mathbf{x}) = \hat{a}_0 + \sum_{k=1}^{K} \hat{a}_k r_k(\mathbf{x}) + \sum_{j=1}^{p} \hat{b}_j l_j(x_j). \tag{12}$$

If the lasso penalized regression is employed, coefficients $\hat{a}$ and $\hat{b}$ are estimated by minimizing:

$$\sum_{i=1}^{N} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}_i) + \sum_{j=1}^{p} b_j l_j(x_{i,j})\right) + \lambda \cdot \left(\sum_{k=1}^{K} |a_k| + \sum_{j=1}^{p} |b_j|\right), \tag{13}$$

where i $(i = 1, \ldots, N)$ denotes an observation in the training dataset. For estimation of coefficients, `pre` employs the `cv.glmnet` function from package **glmnet** (Friedman, Hastie, and Tibshirani 2010). By default, the loss function L is equal to 0.5 times the squared residual for continuous responses and minus the log-likelihood for other response types. Other loss functions (e.g., mean absolute error, misclassification error) may be specified by the user. By default, the penalty parameter λ is set to the value yielding a cross-validated prediction error of one standard error within the minimum, but other values for λ may also be specified by the user.

2.3. Interpretation

Friedman and Popescu (2008) proposed several measures for interpretation of the final prediction rule ensemble. Most are implemented in **pre** and discussed below.

[1]The standard deviation of a typical rule is derived as follows: A rule is a binary $0-1$ coded variable. If we take s to be the proportion of observations to which a rule applies, the variance is given by $f(s) = s(1-s)$, with antiderivative $F(s) = \int s(1-s)\mathrm{d}s = 1/2 \cdot s^2 - 1/3 \cdot s^3$. The average of a function $f(s)$ over its domain $[a, b]$ is given by $1/(b-a)\int_a^b f(s)\mathrm{d}s$. Thus, the average variance of a rule is equal to $1/(1-0)\int_0^1 f(s)\mathrm{d}s = \int_0^1 f(s)\mathrm{d}s = F(1) - F(0) = (1/2 - 1/3) - (0 - 0) = 1/6$, yielding a standard deviation of $\sqrt{1/6} \approx 0.4$.

Importance

To quantify the relative contribution of every base learner to the predictions of the final ensemble, importances can be calculated. Friedman and Popescu (2008) defined the importance of a linear term as:

$$I_j = |\hat{b}_j| \cdot sd(l_j(x_j)), \tag{14}$$

where *sd* denotes the sample standard deviation. Similarly, the global importance of a rule is given by:

$$I_k = |\hat{a}_k| \cdot \sqrt{s_k(1 - s_k)}, \tag{15}$$

where $\sqrt{s_k(1 - s_k)}$ is the sample standard deviation of rule k, which is in turn defined by s_k, the support of rule k in the training data, or the proportion of training observations to which rule k applies:

$$s_k = \frac{1}{N} \sum_{i=1}^{N} r_k(\mathbf{x}_i). \tag{16}$$

The importances in Equations 14 and 15 can be interpreted as the absolute value of regression coefficients, standardized with respect to the base learner. Additional standardization with respect to the outcome variable would yield importances that can be interpreted as standardized regression coefficients. Therefore, **pre** also allows for calculating standardized importances for continuous outcomes. Thus, for linear terms, the standardized importance is given by:

$$I'_j = |\hat{b}_j| \cdot \frac{sd(l_j(x_j))}{sd(y)}. \tag{17}$$

For rules, the standardized importance is given by:

$$I'_k = |\hat{a}_k| \cdot \frac{\sqrt{s_k(1 - s_k)}}{sd(y)}. \tag{18}$$

The total importance of input variable x_j is given by the sum of the importances of the linear term and the rules in which x_j appears (Friedman and Popescu 2008):

$$J_j = I_j + \sum_{x_j \in r_k} \frac{I_k}{c_k}, \tag{19}$$

where c_k is the number of conditions that define rule k. The second term $\sum_{x_j \in r_k} I_k / c_k$ shows that the importance of a rule is distributed equally over the input variables appearing in the rule. When a variable appears more than once in the conditions of a rule, I_k/c_k is multiplied accordingly. The variable importance in Equation 19 can be calculated using standardized or unstandardized importances. In either case, variable importances take values ≥ 0, with higher values indicating a stronger effect on the ensemble's predictions.

Friedman and Popescu (2008) also proposed local importance measures for a selected subregion of the input variable space. These can be obtained by replacing the global standard deviations in Equations 14 and 15 (or 17 and 18 for their standardized counterparts) by the local standard deviations in the subregion. Local variable importances can in turn be calculated by summing the weighted local importances of the base learners in which the variable appears, as in Equation 19.

Partial dependence

The shape of the association between predictor and response variables can be assessed through plotting partial dependence functions. The partial dependence of $F(\mathbf{x})$ on a subset of input variables $S \subset \{1, \dots, p\}$ is defined as the expected value of $F(\mathbf{x})$ over the marginal joint distribution of input variables not in S (i.e., $\mathbf{x}_{\setminus S}$; Friedman 2001; Friedman and Popescu 2008). The partial dependence of $F(\mathbf{x})$ on the subset of predictor variables S can be estimated from the data by:

$$\hat{F}_S(\mathbf{x}_S) = \frac{1}{N} \sum_{i=1}^{N} F(\mathbf{x}_S, \mathbf{x}_{i,\setminus S}), \tag{20}$$

where $\{\mathbf{x}_{i,\setminus S}\}_{i=1}^{N}$ are the training data observations. Taking a subset S of one or two predictor variables allows for plotting the partial dependence of $F(\mathbf{x})$ on $\mathbf{x}_S$.

Interactions

Prediction rules are well suited for capturing interaction effects of input variables. However, non-zero coefficients for rules involving multiple predictor variables in the final ensemble are a necessary but not sufficient condition for the presence of interaction effects. For example, the interaction may be cancelled out by other rules in the ensemble. Or, a rule involving multiple predictor variables may merely reflect main effects of (correlated) predictor variables, instead of interaction(s).

Friedman and Popescu (2008) developed a statistic for assessing whether a predictor variable is involved in interactions with other predictor variables in the model. The underlying rationale is that in the presence of interaction effects, the effects of individual predictor variables are not additive. If an input variable x_j is not involved in interactions with other input variables $\mathbf{x}_{\setminus j}$, then its effect on $F(\mathbf{x})$ is additive, which can then be expressed as:

$$F(\mathbf{x}) = F_j(x_j) + F_{\setminus j}(\mathbf{x}_{\setminus j}), \tag{21}$$

where $F_j(x_j)$ is the partial dependence of $F(\mathbf{x})$ on x_j and $F_{\setminus j}(\mathbf{x}_{\setminus j})$ is the partial dependence of $F(\mathbf{x})$ on $\mathbf{x}_{\setminus j}$, both of which can be estimated from the data using Equation 20. If we assume all partial dependence functions as well as the predictive model $F(\mathbf{x})$ to be centered to have a mean value of zero, the extent to which $F(\mathbf{x})$ deviates from additivity with respect to x_j can be quantified though the following statistic:

$$H_j^2 = \frac{\sum_{i=1}^{N} [F(\mathbf{x}_i) - \hat{F}_j(x_{i,j}) - \hat{F}_{\setminus j}(\mathbf{x}_{i,\setminus j})]^2}{\sum_{i=1}^{N} [F(\mathbf{x}_i)]^2}, \tag{22}$$

where $F(\mathbf{x}_i)$ represent the (centered) model predictions at $\mathbf{x}_i$; $\hat{F}_j(x_{i,j})$ represents the (centered) partial dependence of the predictive model on x_j, evaluated at $x_{i,j}$; $\hat{F}_{\setminus j}(\mathbf{x}_{i,\setminus j})$ represents the (centered) partial dependence of the predictive model on $\mathbf{x}_{\setminus j}$, evaluated at $x_{i,\setminus j}$ and the denominator $\sum_{i=1}^{N} [F(\mathbf{x}_i)]^2$ represents the sample variance of the model's predictions. The statistic H_j^2 measures the fraction of variance of $F(\mathbf{x})$ not captured by the additive effects. It will differ from zero to the extent that x_j is involved in interactions with other input variables.

To assess whether the estimated H_j^2 value significantly differs from zero, a null distribution has to be derived. Friedman and Popescu (2008) suggest the use of a variant of the parametric bootstrap (Efron and Tibshirani 1994) to derive a null distribution for H_j^2. In effect, H_j^2 is repeatedly computed for ensembles fitted on artificial datasets from which interactions are known to be absent. The procedure for generating artificial datasets without interactions and calculating the reference distribution for H_j^2 is described in detail in Friedman and Popescu (2008, Section 8.3).

3. Usage

The basic usage and default settings of function `pre` are as follows:

```
pre(formula, data, weights, family = gaussian, use.grad = TRUE,
  tree.unbiased = TRUE, type = "both", sampfrac = 0.5, maxdepth = 3L,
  learnrate = .01, confirmatory = NULL, mtry = Inf, ntrees = 500,
  tree.control, removeduplicates = TRUE, removecomplements = TRUE,
  winsfrac = 0.025, normalize = TRUE, standardize = FALSE,
  ordinal = TRUE, nfolds = 10L, verbose = FALSE, par.init = FALSE,
  par.final = FALSE, ...)
```

The following arguments are required:

- `formula` provides a symbolic description of the model to be fit.
- `data` specifies a data frame containing the variables in the model.

The following arguments are optional:

- `weights` provides a vector of case weights.
- `family` specifies a GLM-family object or a character string. By default, `family = gaussian` and a single continuous response variable is assumed. Alternatively, for a single binary factor `binomial`, for a count response `poisson`, for a factor with > 2 levels `"multinomial"`, for a multivariate continuous response `"mgaussian"` and for a survival response `"cox"` should be specified. Note that gaussian, binomial and poisson families may be specified as either a GLM-family object or a character string.
- `use.grad` specifies whether a gradient boosting approach should be employed to apply the learning rate. If set to `FALSE`, `glmtree` instead of `ctree` or `rpart` is employed for tree induction.
- `tree.unbiased` specifies whether an unbiased tree generation algorithm should be employed for rule generation. `TRUE` by default, if set to `FALSE`, function `rpart` will be employed for tree induction.
- `type` specifies the type of base learners to be included in the ensemble: `"rules"`, `"linear"` or `"both"`.
- `sampfrac` specifies the fraction of training observations sampled to produce each tree. Values < 1 yield sampling with replacement (sub-sampling), a value of 1 yields sampling with replacement (bootstrap sampling). Alternatively, a sampling function may be supplied, which should take arguments `n` and `weights`.

- `maxdepth` specifies the maximum tree depth and thereby the maximum number of conditions in rules. Should be an integer of length 1 or `ntrees`. Alternatively, a random number generating function may be supplied, which should take argument `ntrees`.
- `learnrate` specifies the value of the learning rate ν to be applied in tree induction.
- `confirmatory` specifies a character vector of confirmatory terms to be included in the final ensemble. No penalty will be applied to the coefficients of these terms, which will yield a non-zero coefficient for the term in the final ensemble.
- `mtry` specifies the number of randomly selected predictor variables for selecting splits in trees. The default `Inf` yields no prior selection of predictor variables.
- `ntrees` specifies the number of trees to be grown for deriving the initial ensemble of trees.
- `tree.control` specifies a list of additional control parameters to be passed to the tree induction algorithm.
- `removeduplicates` specifies whether rules which are identical to earlier generated rules (that is, apply to the same set of observations) should be removed from the initial ensemble.
- `removecomplements` specifies whether rules which are the exact complement of earlier rules (that is, are equal to 1 minus an earlier rule) should be removed from the initial ensemble.
- `winsfrac` specifies the quantiles of the data distribution to be used for winsorizing linear terms. If set to 0, no winsorizing is performed.
- `normalize` specifies whether linear terms should be normalized before estimation of the final ensemble. The default results in every linear term being scaled to have a standard deviation of 0.4, equal to that of a typical rule.
- `standardize` specifies whether all rules and linear terms should be standardized to have unit variance before estimation of the final ensemble.
- `nfolds` specifies the number of folds to be used in calculating the cross-validated error estimates for the possible penalty parameter value for selection of the final ensemble.
- `ordinal` specifies whether ordered factors should be treated as continuous variables for generating rules. The default generally yields simpler rules and computation time.
- `verbose` specifies whether information on model fitting progress should be printed to the command line.
- `par.init` specifies whether parallel computation should be employed for fitting the initial tree ensemble. Note that parallel computation of the initial ensemble will reduce computation time only for (very) large datasets.
- `par.final` specifies whether parallel computation should be employed for selecting the final ensemble.
- `...` specifies additional arguments to be passed to function `cv.glmnet`.

The generated ensemble is returned as an object of class '`pre`', which offers several standard methods and functions for extracting information.

Note that the default settings of pre represent the author's settings of choice, which tend to favor relatively sparse ensembles. The many arguments of function pre allow users to carefully tune accuracy and sparsity of the final ensemble, or to mimic existing tree ensemble approaches for generating rules (e.g., bagging, random forests). Several examples illustrating the tuning options will be provided in Section 4.2. First, the next section will illustrate the functionality of pre in a real-data example with default settings.

4. Examples

4.1. Prediction of depression

To illustrate application of pre, we use a dataset from a study by Carrillo, Rojo, Sánchez-Bernardos, and Avia (2001) which is included in the package. This study examined the extent to which subscales of the NEO personality inventory (NEO-PI; Costa and McCrae 1985) are predictive of depressive symptomatology as measured by the Beck depression inventory (BDI; Beck, Steer, and Carbin 1988). The NEO-PI assesses five major personality dimensions: neuroticism, extraversion, openness to experience, agreeableness and conscientiousness. Each of these dimensions is quantified through six subscale scores and one total score. In the study of Carrillo *et al.* (2001), the NEO-PI and BDI were administered to 112 Spanish respondents. Total scores were calculated for each of the six major dimensions, as well as for each of the neuroticism, extraversion and openness subscales. Respondents' age in years and sex were also included in the dataset. Further information about the study and sample is provided in Carrillo *et al.* (2001).

First, we load the package and data:

```
R> library("pre")
R> data("carrillo", package = "pre")
```

We derive a prediction rule ensemble using function pre. As rule derivation and selection of the final ensemble depends on random sampling of the training data, we first set the random seed:

```
R> set.seed(42)
R> carrillo.ens <- pre(bdi ~ ., data = carrillo)
R> carrillo.ens

Final ensemble with cv error within 1se of minimum:
  lambda =  0.7779287
  number of terms = 14
  mean cv error (se) = 37.05145 (6.212951)

  cv error type : Mean-Squared Error

         rule  coefficient              description
  (Intercept)   8.93985818                        1
       rule80   2.40103417     n4 > 15 & open4 <= 13
       rule88  -1.35678309     ntot <= 109 & e6 > 15
       rule97  -1.20824458     n2 <= 16 & open4 > 10
       rule18  -1.02385159  ntot <= 109 & etot > 101
        rule1  -0.99590845                  n3 <= 17
       rule12  -0.58728116                  n3 <= 22
       rule42  -0.51972664     n6 <= 19 & open4 > 12
       rule86  -0.26280878        n2 <= 16 & e6 > 14
       rule66   0.25137293   open4 <= 13 & ntot > 82
           n3   0.17522150         2 <= n3 <= 30.225
      rule105  -0.14432622     n2 <= 16 & open5 > 11
       rule30  -0.06473571    ntot <= 109 & n4 <= 14
       rule46  -0.05753864                  n1 <= 20
       rule40  -0.04878180                  n6 <= 19
```

The printed result reports that the final ensemble with cross-validated error within one standard error above the minimum was selected. This is the default employed by `print` and other functions in **pre**. Alternatively, the `penalty.parameter.val` argument can be set to `"lambda.min"` (λ value yielding the minimum cross-validated error), or a numeric value > 0. Note that the reported cross-validated error is calculated using the same data as used for deriving the prediction rules and likely provides an overly optimistic estimate of future prediction error. Performing full cross-validation will yield a more honest estimate of prediction error, for which we will use function `cvpre` later on in the example.

The printed result shows each of the selected base learners in the final ensemble with the corresponding coefficients. Base learners with an estimated coefficient of 0 are omitted from this output, by default. The first column (`rule`) indicates the type of base learner: a rule (e.g., `rule80`) or linear term (e.g., `n3`). The `description` column lists the conditions for rules and the winsorizing points for linear terms, if winsorizing was performed (note that `n3` was winsorized with the default value of $\beta = 0.025$). The first rule shows that observations with a higher value of `n4` (i.e., n4 > 15) and a lower value of `open4` (i.e., open4 ≤ 13) have an expected BDI score 2.4 higher than observations that do not match these conditions.

The results indicate that depressive symptomatology is mostly predicted by the neuroticism (sub)scales. This is not surprising, given that these scales were specifically constructed to assess emotional adjustment and (in)stability, with higher scores indicating higher proneness to psychological distress. The `ntot` variable reflects the total score on this scale, while the `n1`, `n2`, `n3`, `n4` and `n6` variables reflect the scores on the anxiety, anger & hostility, depression, self-consciousness and vulnerability subscales, respectively. Furthermore, variable `e6` represents a subscale of the extraversion scale, reflecting proneness to the experience of positive emotions. The `open4` and `open5` variables represent subscales of the openness to experience scale, capturing openness to actions and ideas, respectively.

We can obtain the estimated (zero and non-zero) coefficients for the base learners in the final ensemble using the `coef` method (results not shown for space considerations):

```
R> coef(carrillo.ens)
```

We can obtain predictions for (new) observations using the `predict` method (results not shown for space considerations):

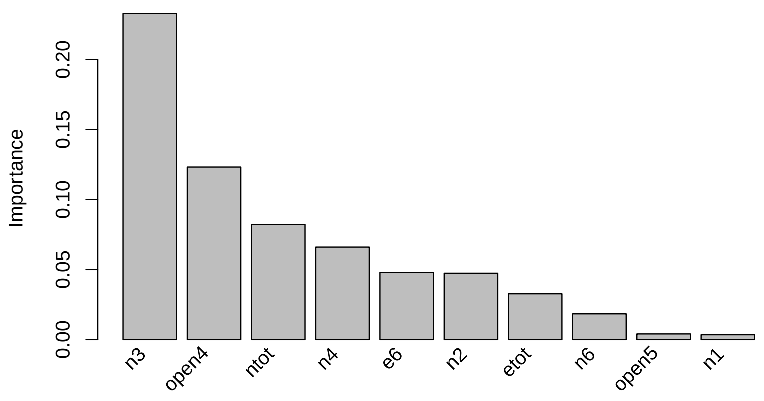

Figure 2: Input variable importances for the prediction rule ensemble for predicting depression.

```
R> predict(carrillo.ens, newdata = carrillo)
```

We can obtain variable and base learner importances using the `importance` function. By default, importances are calculated over all training observations, but the `importance` function also allows for obtaining local importances calculated over a subset of the training data, through specification of the `global` and `quantprobs` arguments. To aid in interpretation, we request standardized importances in this example, so we can interpret the base learner importances as the absolute value of standardized multiple regression coefficients. Also, we restrict the number of decimal places in the results by specifying the `round` argument:

```
R> imps <- importance(carrillo.ens, standardize = TRUE, round = 4L)
```

Figure 2 displays the input variable importances. The two most important input variables for predicting depressive symptoms are a neuroticism (`n3`) and an openness subscale (`open4`). In addition to a plot of input variable importances, `importance` returns a list of base learner and variable importances, respectively (i.e., `baseimps` and `varimps`; only the first three rows of the former are shown here):

```
R> imps$baseimps[1:3, ]
```

```
    rule            description    imp coefficient     sd
1     n3     2 <= n3 <= 30.225 0.1476      0.1752 6.6030
2 rule80 n4 > 15 & open4 <= 13 0.1281      2.4010 0.4183
3 rule88 ntot <= 109 & e6 > 15 0.0806     -1.3568 0.4656
```

We can plot (a subset of) the final ensemble using the `plot` function. Below, `standardize = TRUE` is specified so that the importances in the plots are standardized, `nterms = 6` so that only the six most important base learners will be plotted, `plot.dim = c(2, 3)` so that the rules will be plotted in two rows and three columns and `cex = 0.7` to scale the size of node and edge labels to fit the plot size:

```
R> plot(carrillo.ens, nterms = 6, plot.dim = c(2, 3), standardize = TRUE,
+    cex = 0.7)
```

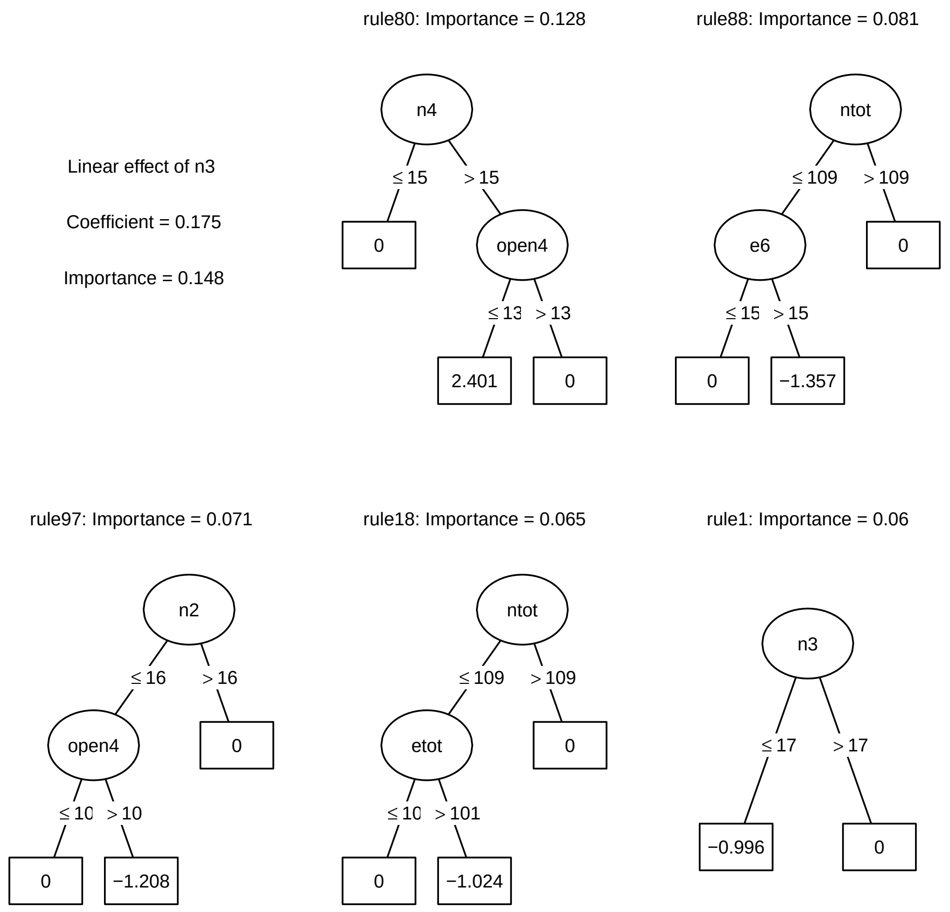

Figure 3: The six most important base learners in the prediction rule ensemble for predicting depression.

Figure 3 displays the six most important base learners in the ensemble. The most important base learner is a linear term, `n3`. The second most important base learner is a rule involving `n4` and `open4`. Together, these base learners indicate a positive association of depressive symptomtomatology with neuroticism, and a negative association with extraversion and openness.

To further inspect the shape of the effect of individual input variables, we can obtain a partial dependence plot using the `singleplot` function:

```
R> singleplot(carrillo.ens, varname = "ntot")
```

Figure 4 displays the partial dependence plot, which indicates a monotonically increasing, rather stepwise association between the neuroticism scale and depressive symptomatology. To inspect the combined association between a pair of predictor variables and the response, we can employ the `pairplot` function:

```
R> pairplot(carrillo.ens, varnames = c("n4", "open4"),
+    col = grey(seq(1, 0.4, by = -0.01)))
```

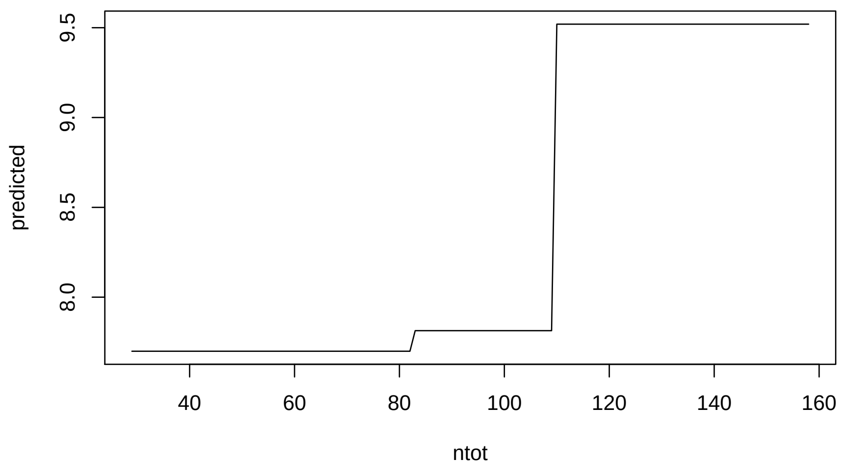

Figure 4: Partial dependence plot of the response variable on input variable `ntot`.

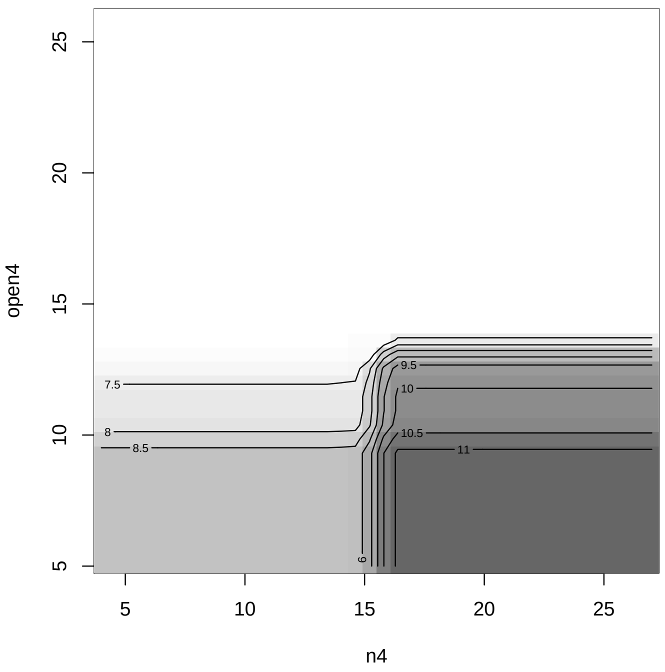

Figure 5: Partial dependence plot of the response variable on input variables `n4` and `open4`.

By default, `pairplot` employs a plotting color sequence going from yellow (lower values) to red (higher values). Here, a sequence from light to dark grey has been specified through the `col` argument. Figure 5 displays the partial dependence of the depression variable on `n4` and `open4`. The plot indicates that depressive symptomatology increases with increasing values of the neuroticism subscale, and decreases with increasing values of the openness subscale.

The pattern revealed by the partial dependence may reflect an interaction, or two main effects. If we want to assess and test the presence of interaction effects, we can employ the `interact` and `bsnullinteract` functions. The latter fits PREs on bootstrapped null-interaction datasets, that is, bootstrap sampled datasets from which interactions are known

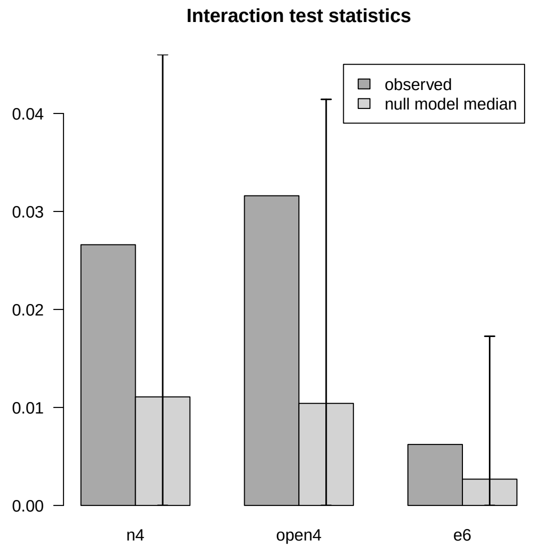

Figure 6: Interaction test statistics for input variables `n4`, `open4` and `e6`. The darker grey bars represent interaction strengths of the fitted ensemble; the lighter grey bars represent the median interaction strength in the null interaction models, with the error bars indicating the 0.05 and 0.95 quantiles of the distribution.

to be absent. Generating these null-interaction models is computationally intensive, therefore `bsnullinteract` generates ten null-interaction models, by default. To obtain a more precise estimate of the null distribution of interaction test statistics, we increase the number of generated null-interaction ensembles by specifying the `nsamp` argument (which will take substantially longer to compute). Because generating the null-interaction ensembles requires random sampling and permutation of the training data, we first set the random seed:

```
R> set.seed(43)
R> nullmods <- bsnullinteract(carrillo.ens, nsamp = 100)
```

Next, we obtain interaction test statistics for both the fitted and null-interaction models:

```
R> int.carrillo <- interact(carrillo.ens, nullmods = nullmods,
+    varnames = c("n4", "open4", "e6"))
```

Figure 6 displays the interaction test statistics for `n4`, `open4` and `e6`. The darker bars represent the observed interaction strengths in the fitted ensemble. The lighter bars represent the median interaction strength in the null-interaction models, with the error bars indicating the 0.05 and 0.95 quantiles of the distribution. The plot indicates that none of the three specified predictor variables are involved in interactions.

Finally, using the `cvpre` function, we estimate the out-of-sample prediction error of the fitted ensemble through full k-fold cross-validation. By default, the number of folds is set to ten. As cross-validation involves random sampling of observations, we first set the random seed:

```
R> set.seed(44)
R> cv.carrillo <- cvpre(carrillo.ens)
```

```
$MSE
      MSE        se
47.025425  8.065651

$MAE
      MAE        se
5.1876512 0.4256812
```

The printed results show the mean squared error (MSE) and mean absolute error (MAE) and their respective standard errors. For classification, mean squared and mean absolute error loss would be returned. With the code above, these accuracy estimates are also saved in `cv.carrillo$accuracy`. In addition, `cv.carrillo$cvpreds` contains the cross-validated predictions, which can be used to calculate alternative indices of predictive accuracy.

4.2. Customizing rule induction and model selection

The default settings of `pre` employ a sub-sampling (i.e., `sampfrac = 0.5`) and boosting (i.e., `learnrate = 0.01`) approach. As in most tree boosting algorithms, maximum tree depth is also limited by default (i.e., `maxdepth = 3L`). Note that, in addition to the maximum tree depth, the `ctree` function simultaneously employs an additional stopping criterion: by default an α level of 0.05 for split selection is used. That is, no split is performed when all of the partitioning variables show p values > 0.05. The latter can be adjusted through specification of the `tree.control` argument.

The following examples aim to illustrate how several well-known tree ensemble approaches can be mimicked. The resulting ensembles are not shown here, but will likely be more complex than ensembles generated with the default settings.

Bagging

A bagging approach can be mimicked by setting `sampfrac = 1` (to employ bootstrap sampling), specifying a learning rate of 0 and applying no a-priori restrictions on tree size (i.e., employing unpruned trees):

```
R> set.seed(42)
R> car.ens.bag <- pre(bdi ~ ., data = carrillo, sampfrac = 1,
+    maxdepth = Inf, learnrate = 0,
+    tree.control = partykit::ctree_control(alpha = 1))
```

Note that increasing the value of `maxdepth` will increase both computation time and complexity of the final ensemble.

Random forest

A random forest approach can be mimicked by additionally restricting the number of predictor variables considered for selecting each split through the `mtry` argument:

```
R> set.seed(42)
R> car.ens.ranfor <- pre(bdi ~ ., data = carrillo, sampfrac = 1,
+    maxdepth = Inf, learnrate = 0,
+    tree.control = partykit::ctree_control(alpha = 1),
+    mtry = ceiling(sqrt(ncol(carrillo))))
```

Additionally setting `tree.unbiased = FALSE` would employ the `rpart` implementation of the CART algorithm, which would most closely resemble the original bagging and random forest approaches. Note that this will generally yield more complex ensembles.

Original RuleFit approach

The default settings of the original RuleFit implementation can be mimicked by employing the CART algorithm, and letting the sampling fraction and number of cross-validation folds depend on the number of effective observations, which is equal to the sample size for regression. Furthermore, the maximum tree depth should be determined by a random number generating function, allowing for varying tree depths (Friedman and Popescu 2008, 2012). Such a function can be generated using the `maxdepth_sampler` function, which by default samples from a distribution with an average maximum tree depth of two (as in the original RuleFit implementation):

```
R> neff <- nrow(carrillo)
R> set.seed(42)
R> car.ens.rulef <- pre(bdi ~ ., data = carrillo, tree.unbiased = FALSE,
+    maxdepth = maxdepth_sampler(),
+    sampfrac = min(1, (11 * sqrt(neff) + 1) / neff),
+    nfolds = round(min(20, max(0, 5200 / neff - 2))))
```

Furthermore, to fully mimic the original RuleFit approach, the minimum cross-validated error criterion should be employed to obtain the final ensemble:

```
R> print(car.ens.rulef, penalty.par.val = "lambda.min")
```

Regularized single tree

An extremely sparse PRE would take the nodes of only a single tree as the initial ensemble. The predictions of the resulting ensemble would represent regularized node means of the single (pruned) tree. When only a single tree is grown, bootstrap or sub-sampling may not improve predictive accuracy, so we first define a custom sampling function, which returns all indices of the training observations:

```
R> samp_func <- function(...) 1:nrow(carrillo)
```

Then we specify only a single tree to be grown and no linear terms to be included:

```
R> car.ens.tree <- pre(bdi ~ ., data = carrillo, ntrees = 1,
+    type = "rules", sampfrac = samp_func)
```

Regularization of the final ensemble

Through the ellipsis (`...`) `pre` allows for passing additional arguments to the `cv.glmnet` function. For example, this allows for specifying the elastic net mixing parameter α. By default, the lasso penalty is employed (i.e., `alpha = 1`), but specifying `alpha = 0` would yield the ridge penalty, and values $0 < \alpha < 1$ would yield the elastic net penalty. An unpenalized solution can be obtained by supplying a pre-specified range for the penalty parameter λ, including 0. Such an approach will likely yield sub-optimal sparsity and accuracy, but may in rare cases be preferred over a penalized solution. Note that the `cv.glmnet` function only permits specification of multiple λ values, so we have to specify at least two λ values:

```
R> set.seed(42)
R> car.ens.unp <- pre(bdi ~ ., data = carrillo, lambda = c(0, 1))
```

To obtain the unpenalized solution, all methods and functions applied to the resulting object need to specify `penalty.par.val = 0`:

```
R> print(car.ens.unp, penalty.par.val = 0)
```

Finally, to employ a different loss function than the default squared-error or log-likelihood criterion, `type.measure` can be set to, for example, `"class"` (for misclassification error) or `"mae"` (for mean absolute error):

```
R> set.seed(42)
R> car.ens.mae <- pre(bdi ~ ., data = carrillo, type.measure = "mae")
```

5. Empirical evaluation

5.1. Method

Datasets

Four benchmark datasets were employed to compare the methods: two regression datasets (`BostonHousing` and `Ozone`) and two classification datasets (`BreastCancer` and `Ionosphere`), each obtained from the UCI Machine Learning Repository (Dua and Karra Taniskidou 2017) through the **mlbench** package (version 2.1-1; Leisch and Dimitriadou 2012). Only complete observations were included in the analyses. Specifically, no observations were removed from the `BostonHousing` dataset. From the `BreastCancer` dataset, 16 cases were removed due to missing values. From the `Ionosphere` dataset, one variable was removed due to zero variance. From the `Ozone` dataset, a categorical variable with a large number of categories was removed to reduce computation time (i.e., day of month), one variable was removed due to a large number of missing values (temperature at El Monte, California) and a total of 36 cases were removed due to missing values. Table 1 provides the resulting total sample sizes and numbers of predictor variables for each dataset.

Model fitting procedures

The performance of `pre` was compared with that of random forests, single regression trees, lasso penalized linear regression and the original RuleFit implementation. All analyses were performed in R (version 3.6.1; R Core Team 2019). To fit PREs, function `pre` from package

Dataset	Response	p	N
`BostonHousing`	numeric	13	506
`BreastCancer`	binary factor	9	683
`Ionosphere`	binary factor	33	351
`Ozone`	numeric	10	330

Table 1: Benchmark datasets used for comparing performance; p refers to the total number of predictor variables in the dataset, N refers to the total sample size.

pre (version 1.0.0; Fokkema and Christoffersen 2020) was employed. In addition, to fit PREs with the original RuleFit implementation, **RuleFit** version 3 (Friedman and Popescu 2012) for Windows was obtained from `https://statweb.stanford.edu/~jhf/R_RuleFit.html` and function `rulefit` was employed. To fit single trees, function `ctree` from package **partykit** (version 1.2-6; Hothorn and Zeileis 2015) was used to fit conditional inference trees, and function `rpart` from package **rpart** (version 4.1-15; Therneau *et al.* 2019) was used to fit CART trees. To fit random forests, function `cforest` from package **partykit** was used to fit random forests based on conditional inference trees, and function `randomForest` from package **randomForest** (version 4.6-14; Liaw and Wiener 2002) was used to fit random forests based on CART trees. To fit lasso penalized linear regression models, function `cv.glmnet` from package **glmnet** (version 3.0-2; Friedman *et al.* 2010) was used.

All analyses employed default settings, with two exceptions: For fitting PREs, maximum rule length was set to 4, instead of the default value of 3. Furthermore, trees fitted with **rpart** were pruned using the 1 standard error criterion. Note that careful tuning of parameter settings using cross-validation approaches would likely yield higher predictive accuracy for all methods.

For functions `rulefit` and `pre`, selection of the final ensembles was performed using two different criteria. For each, a final ensemble was obtained through lasso regression, with the value of λ set to minimize squared error loss based on cross-validation. In addition, for `pre`, a sparser ensemble was obtained through employing a λ value yielding squared error loss within one standard error above the minimum (which is the default in package **pre**). For `rulefit`, a sparser ensemble was obtained through forward stepwise regression for numeric outcomes, and through forward stagewise regression with variable entry order determined by lasso regression for binary outcomes. For `pre`, the number of cross-validation replications was set to the default value of 10; for `rulefit`, the default was also employed, which is a function of sample size, yielding 6 to 14 replications in the current analyses.

Assessment of performance

To assess performance, the bootstrap cross-validation design for benchmark experiments with real-world data of Hothorn, Leisch, Zeileis, and Hornik (2005) was employed. From each dataset, 250 bootstrap samples were drawn. Each bootstrap sample was used for training and predictive accuracy was subsequently assessed using the test observations that were not included in the bootstrap sampled training data. Predictive accuracy was quantified through calculating mean squared error (MSE) in regression problems and the area under the receiver operating curve (AUC) value in classification problems. Interpretability was assessed through counting the number of predictor variables, for the `BostonHousing` and `Ionosphere` data. As

	BostonHousing	BreastCancer	Ionosphere	Ozone	Mean rank
randomForest	11.75 (3.41)	0.993 (0.003)	0.979 (0.011)	17.12 (2.22)	1.25
cforest	16.47 (4.46)	0.993 (0.004)	0.971 (0.014)	17.78 (2.37)	3.50
pre_1se	14.06 (3.69)	0.991 (0.005)	0.961 (0.018)	17.72 (2.67)	3.50
rulefit_min	12.90 (3.29)	0.990 (0.006)	0.961 (0.017)	20.19 (3.02)	4.00
pre_min	13.35 (3.48)	0.991 (0.005)	0.960 (0.019)	18.85 (2.87)	4.25
lasso	27.58 (5.12)	0.995 (0.002)	0.902 (0.032)	20.31 (2.27)	5.75
ctree	21.68 (4.94)	0.975 (0.011)	0.908 (0.032)	24.88 (4.16)	7.00
rulefit_FS	15.35 (4.46)	0.945 (0.016)	0.883 (0.032)	24.32 (4.22)	7.50
rpart	25.34 (5.74)	0.948 (0.018)	0.892 (0.037)	26.01 (4.07)	8.25

Table 2: Predictive accuracy for all methods and datasets. Values represent averages over 250 bootstrap samples, with standard deviations in parentheses. For the `BostonHousing` and `Ozone` data, mean squared error (MSE) was calculated; for the `BreastCancer` and `Ionosphere` data, the area under the receiver operating characteristic curve (AUC) values were calculated. Mean rank represents the ranking of the algorithms from highest to lowest predictive accuracy, averaged over the four datasets.

it was assumed the two random forest methods and the two single tree methods would yield similar complexities, respectively, only the number of variables for `cforest` and `ctree` were counted. For the `BreastCancer` and `Ozone` datasets, the total number of base learners with non-zero coefficients (i.e., the number of terms) were counted. The number of terms were only evaluated for `pre` and `rulefit`. Finally, computation time in seconds was recorded for every fitted model.

5.2. Results

Table 2 presents average predictive accuracies in each of the benchmark datasets. Overall, function `randomForest` ranked highest in terms of predictive accuracy, followed by `cforest` and `pre` (employing the 1 standard-error criterion), followed by `rulefit`, followed by `pre` (each of the latter two employing the minimum cross-validation error criterion). Overall, `pre` showed higher accuracy than `rulefit`. Of note, `rulefit` employing the minimum cross validation error criterion showed substantially better accuracy than `rulefit` employing forward selection. The latter model showed predictive accuracy comparable to single trees.

Figure 7 depicts the distributions of predictive accuracy across the four benchmark datasets. The boxplots indicate that the most accurate methods also show the least variation in predictive accuracy. Figure 8 depicts the fitted models' complexities. The random forests always included all predictor variables, providing the least sparse solutions. After random forests, lasso regression provided the least sparse solutions, followed by PREs selected using lasso regression. The sparsest solutions were provided by singles trees (ctrees) and rulefit ensembles with forward stepwise selection. Comparing the complexity of PRE methods yields a similar pattern in all datasets: `rulefit` with forward selection provided the sparsest PREs, followed by `pre` with the $1 - \text{SE}$ rule, followed by `pre` with minimum cross validated error, followed by `rulefit` with minimum cross validated error. Taken together with the findings on predictive accuracy, these results indicate that `pre` may provide a better trade-off between accuracy and interpretability than `rulefit`.

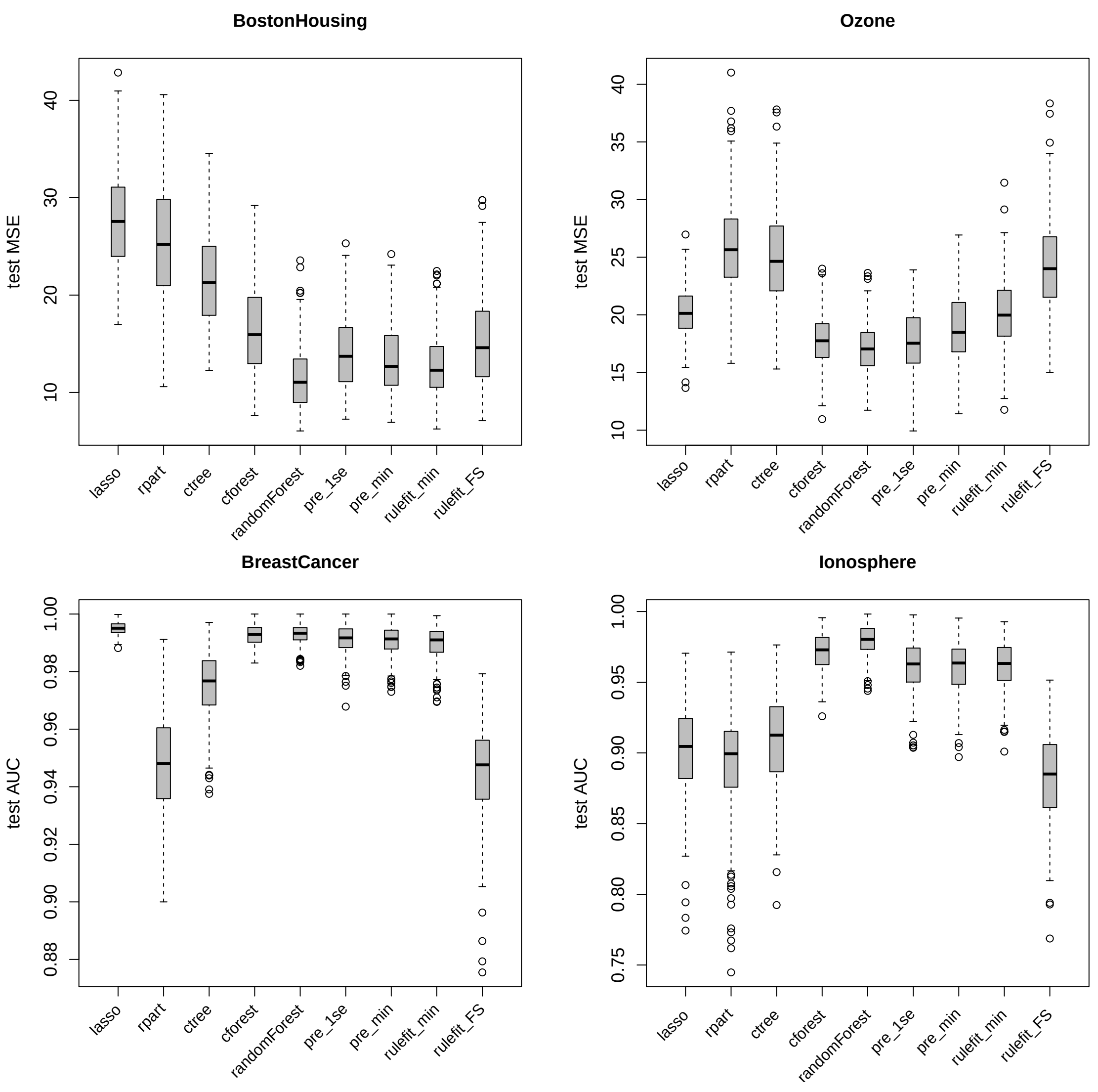

Figure 7: Predictive accuracy for the different algorithms in 250 bootstrap samples of the benchmark datasets. Three AUC values for **rpart** were winsorized to 0.90 in the `BreastCancer` dataset to improve the graphical presentation. MSE = mean squared error; AUC = area under the receiver operating characteristic curve; FS = forward selection.

Figure 9 depicts the computation time distributions for all methods. Note that for `pre`, only a single boxplot is depicted, because the $1-\mathrm{SE}$ and minimum cross validated error solutions are obtained from the same fitted model. Figure 9 shows a clear computational disadvantage for `pre`, with an average computation time of 18 seconds. Most computation time for `pre` is spent on the fitting of ctrees, which is computationally more demanding than fitting CART trees. This can also be observed in the computation time differences between `ctree` (0.066 seconds, on average) and `rpart` (0.013 seconds, on average), and between `cforest` (12.1 seconds, on average) and `randomForest` (0.3 seconds, on average). The `rulefit` function also employs CART trees and yielded average computation times of 1.5 seconds (with forward selection) and 2.6 seconds (with lasso selection).

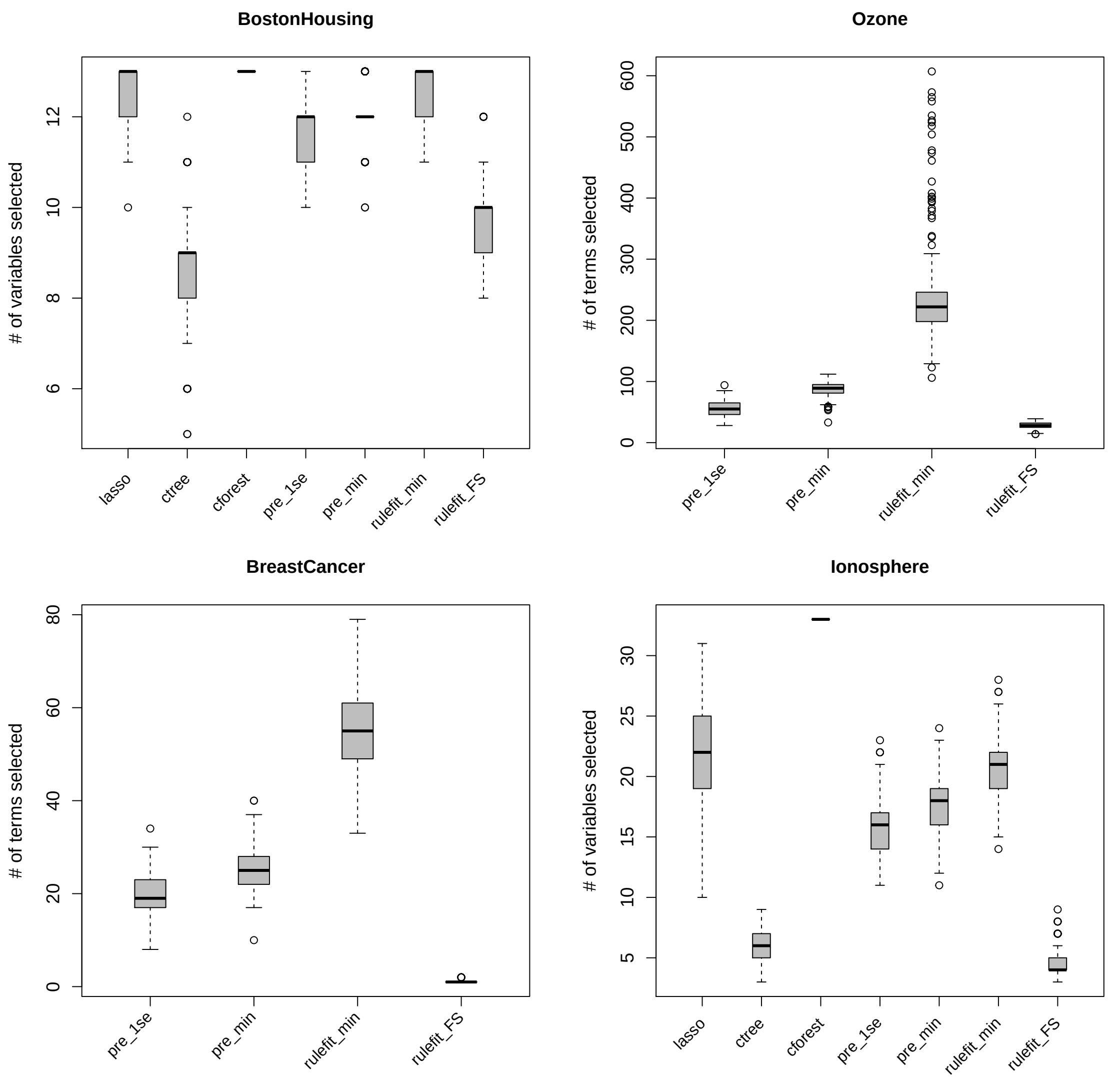

Figure 8: Distribution of the number of predictor variables or number of terms selected in 250 bootstrap samples of the benchmark datasets.

6. Comparison with other packages

A number of algorithms and software packages for fitting PREs have been developed over the last years. An extensive empirical comparison is outside the scope of the current paper, but several packages are discussed and compared below. Following Frank and Witten (1998), we distinguish between two strategies for generating rules: Indirectly, through transforming the nodes of one or more decision trees to a set of rules, and directly, through for example a sequential covering approach (Fürnkranz 1999).

The indirect approach to rule generation is employed in the method of Friedman and Popescu (2008). The **RuleFit** program (Friedman and Popescu 2012), which is written in Fortran and can be executed through an R interface, provides a fast implementation of the method, which was also observed in Section 5. In addition, several R packages implement a two-step approach

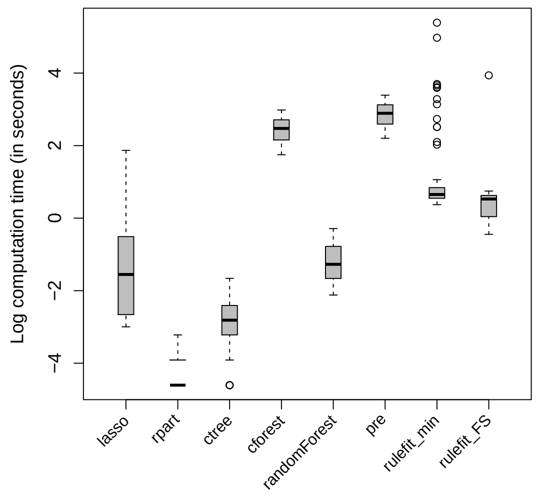

Figure 9: Distribtuions of the log of computation time (in seconds) across all bootstrap sampled datasets.

similar to that of Friedman and Popescu (2008): **inTrees** (Deng 2018), **horserule** (Nalenz and Villani 2018) and **nodeHarvest** (Meinshausen 2010) also generate rules from the nodes of tree ensembles, after which weights or coefficients are estimated to construct a final ensemble. For this second step, each package employs a different approach: Package **inTrees** uses a sequential covering approach (Fürnkranz 1999) to select a sparse final ensemble of rules. Package **horserule** estimates rule coefficients using a Bayesian linear model with horseshoe prior (Carvalho, Polson, and Scott 2010). Although the horseshoe estimator has been found to yield better predictive accuracy than lasso regression, it does not enforce a sparse solution as it does not shrink coefficients to a value of zero. Finally, package **nodeHarvest** obtains node weights through solving a quadratic programming problem with linear inequality constraints, yielding non-zero weights for only a (small) subset of nodes (Meinshausen 2010).

The main advantage of the node harvest approach is that it does not require selection of a tuning parameter, as estimation with the lasso does. Also, node harvest predictions are given by weighted node means, which may aid interpretation: If an observation falls only into a single node, the prediction is the average response among the training observations in that node. In contrast, the coefficients in Equation 12 are shrunken towards zero and cannot be interpreted as node means. On the other hand, the lasso regression model in Equation 13 can more easily be extended to include linear (and other) functions of predictor variables.

C5.0 (Quinlan 1993; RuleQuest 2017) also employs an indirect approach to rule learning. **C5.0** performs classification only, is written in C and is available as a standalone executable file. Alternatively, package **C50** (Kuhn and Quinlan 2020) provides an R interface. The predecessor **C4.5** is described in detail in (Quinlan 1993), while the documentation on the implementation of **C5.0** is limited (RuleQuest 2017), but Kuhn and Johnson (2013) provide a rather complete description. **C5.0** allows for generating PREs from the nodes of a single tree or a boosted tree ensemble. In the former case, a set of rules is derived from the nodes of a single tree and simplified through pruning and deletion of rules, so as to minimize prediction error. Predictions for new observations are generated by a weighted majority vote of the rule ensemble. When boosting is applied, observation weights are adjusted based on the current classification error in every iteration. Predictions for new observations are then given by the average of the predicted class probability of each of the rule sets. Although **C5.0** tree ensembles rank among the most accurate classifiers, **C5.0** rule ensembles have been found to perform less well (e.g., Fernández-Delgado, Cernadas, Barro, and Amorim 2014).

Weka's (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009) sub-package **classifiers.rules** implements several algorithms for deriving PREs: `JRip` (implementing the RIPPER algorithm of Cohen 1995), `M5Rules` (Quinlan 1992; Holmes, Hall, and Frank 1999) and `PART` (Frank and Witten 1998). `M5Rules` builds a PRE for regression through a sequential covering approach; it builds a tree in every iteration and takes the best node as a rule. `PART` employs the same approach for building a PRE for classification. `JRip` generates rules directly through a sequential covering approach. As the sequential covering approach is likely to be outperformed by boosting (e.g., Cohen and Singer 1999; Dembczyński *et al.* 2010), these algorithms will not be further discussed here.

Another algorithm that generates rules directly is ENDER (Dembczyński *et al.* 2010), which provides a very general framework for generating boosted PREs. It is implemented in **RegENDER** (Dembczyński, Kotłowski, and Słowiński 2008), which is written in Java and can be executed from **Weka**. ENDER allows users to select from a range of loss functions and regularization methods, thereby also encompassing classification rule ensemble learners like SLIPPER (Cohen and Singer 1999) and lightweight rule induction (Weiss and Indurkhya 2000). Notably, Dembczyński *et al.* (2010) report that predictive accuracy is hardly affected by the choice of loss function, but substantially affected by the regularization methods employed. They found regularization through application of a learning rate, resampling of observations and calculating coefficients on the complete training data instead of sub-samples to yield improved accuracy of the final ensemble.

The main difference between **pre** and other packages employing an indirect approach to rule generation is in the tree induction algorithm. The results in Section 5 indicate that conditional inference trees provide equal or better predictive accuracy than CARTs. Also, Schauerhuber, Zeileis, Meyer, and Hornik (2007) found conditional inference trees to yield lower complexity than C4.5 trees, but higher complexity than CARTs. The current findings on the lower complexity of `pre` compared to `rulefit` may be due to `ctree`'s default stopping criterion, where splitting is halted when the null hypothesis of independence between any of the input variables and the response cannot be rejected in a node. The `rulefit` function does not employ such a data-driven stopping criterion. The current results suggest that `ctree`'s stopping criterion may improve both sparsity and predictive accuracy of the final ensemble.

Finally, **pre** and **RegENDER** are similar in that they apply regularization through boosting, sampling and global estimation of rule coefficients. The main difference between the two packages lies in the rule-generation approach employed. Dembczyński *et al.* (2010) note that the main advantage of their approach is that the rules are constructed directly based on impurity measures. Specifying a minimum value for improvement of the impurity measure then yields a natural stopping criterion for building rules. However, the unbiased tree induction algorithms employed by **pre** also provide a natural stopping criterion, because the splitting is based on statistical testing. In addition, the maximum number of conditions that may appear in rules, as specified by the `maxdepth` argument of `pre`, provides an additional stopping criterion for building rules.

7. Conclusion

The current paper presented function `pre` from R package **pre**, which allows for deriving prediction rule ensembles for (multivariate) continuous, binary, multinomial, count and survival outcomes. The fitting procedures and measures for interpretation as implemented in package **pre** were discussed. Using an example dataset on the prediction of depressive symptomatology, a rule ensemble was derived and inspected. In four benchmark datasets, the performance of function `pre` was compared with the original RuleFit implementation, random forests, single trees and lasso penalized regression models. Results indicated that `pre` provided slightly better accuracy than the original RuleFit implementation. Furthermore, `pre` provided accuracy similar to random forests, while providing substantially lower complexity than both random forests and RuleFit. The lower complexity of `pre` is likely due to the use of unbiased recursive partitioning methods, which do not have a preference for variables with many possible splitting values and employ a statistical criterion for split selection. Although `pre` provided a better trade-off between complexity and accuracy, it also yielded the longest computation times. Future developments will focus on reducing computation time, improving modularity and extending the range of methods that can be employed for selecting the final ensemble.

Acknowledgments

The author would like to thank Benjamin Christoffersen for his contributions to the development of package **pre**. Also, the author would like to thank Henk Kelderman for commenting on an earlier version of this manuscript. Part of the work in this publication was carried out during a research visit to the University of Zurich, supported through an International Short Visit grant from the Swiss National Science Foundation (IZK0Z1_175531).

References

Beck AT, Steer RA, Carbin MG (1988). "Psychometric Properties of the Beck Depression Inventory: Twenty-Five Years of Evaluation." *Clinical Psychology Review*, **8**(1), 77–100. doi:10.1016/0272-7358(88)90050-5.

Breiman L (1996). "Heuristics of Instability and Stabilization in Model Selection." *The Annals of Statistics*, **24**(6), 2350–2383. doi:10.1214/aos/1032181158.

Breiman L, Friedman J, Olshen R, Stone C (1984). *Classification and Regression Trees*. Wadsworth, New York.

Carrillo JM, Rojo N, Sánchez-Bernardos ML, Avia MD (2001). "Openness to Experience and Depression." *European Journal of Psychological Assessment*, **17**(2), 130–136. doi:10.1027//1015-5759.17.2.130.

Carvalho CM, Polson NG, Scott JG (2010). "The Horseshoe Estimator for Sparse Signals." *Biometrika*, **97**(2), 465–480. doi:10.1093/biomet/asq017.

Cohen WW (1995). "Fast Effective Rule Induction." In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123. Morgan Kaufmann Publishers, San Mateo.

Cohen WW, Singer Y (1999). "A Simple, Fast, and Effective Rule Learner." In *Proceedings of the National Conference on Artificial Intelligence*, pp. 335–342. John Wiley & Sons Ltd, New York.

Costa PT, McCrae RR (1985). *The NEO Personality Inventory*. Psychological Assessment Resources, Odessa.

De Bin R, Janitza S, Sauerbrei W, Boulesteix AL (2016). "Subsampling versus Bootstrapping in Resampling-Based Model Selection for Multivariable Regression." *Biometrics*, **72**(1), 272–280. doi:10.1111/biom.12381.

Dembczyński K, Kotłowski W, Słowiński R (2008). "Solving Regression by Learning an Ensemble of Decision Rules." In *International Conference on Artificial Intelligence and Soft Computing, 2008*, pp. 533–544. Springer-Verlag, Heidelberg, Germany.

Dembczyński K, Kotłowski W, Słowiński R (2010). "ENDER: A Statistical Framework for Boosting Decision Rules." *Data Mining and Knowledge Discovery*, **21**(1), 52–90. doi:10.1007/s10618-010-0177-7.

Deng H (2018). ***inTrees**: Interpret Tree Ensembles.* R package version 1.2, URL https://CRAN.R-project.org/package=inTrees.

Dietterich T (2000). "Ensemble Methods in Machine Learning." In J Kittler, F Roli (eds.), *Multiple Classifier Systems*, pp. 1–15. Springer-Verlag, Berlin.

Dua D, Karra Taniskidou E (2017). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml/.

Efron B, Tibshirani RJ (1994). *An Introduction to the Bootstrap*. Chapman & Hall/CRC, New York.

Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014). "Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?" *Journal of Machine Learning Research*, **15**(1), 3133–3181.

Fokkema M, Christoffersen B (2020). ***pre**: Prediction Rule Ensembles.* R package version 1.0.0, URL https://CRAN.R-project.org/package=pre.

Fokkema M, Smits N, Kelderman H, Penninx BWJH (2015). "Connecting Clinical and Actuarial Prediction with Rule-Based Methods." *Psychological Assessment*, **27**(2), 636–644. doi:10.1037/pas0000072.

Frank E, Witten IH (1998). "Generating Accurate Rule Sets without Global Optimization." In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 144–151. Morgan Kaufmann Publishers, San Mateo.

Friedman J (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **29**(5), 1189–1232. doi:10.1214/aos/1013203451.

Friedman JH, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.

Friedman JH, Popescu BE (2003). "Importance Sampled Learning Ensembles." *Technical report*, Stanford University. URL `http://www-stat.stanford.edu/~jhf/ftp/isle.pdf`.

Friedman JH, Popescu BE (2008). "Predictive Learning via Rule Ensembles." *The Annals of Applied Statistics*, **2**(3), 916–954. `doi:10.1214/07-aoas148`.

Friedman JH, Popescu BE (2012). "**RuleFit** with R." URL `http://www-stat.stanford.edu/~jhf/R-RuleFit.html`.

Fürnkranz J (1999). "Separate-And-Conquer Rule Learning." *Artificial Intelligence Review*, **13**(1), 3–54. `doi:10.1023/a:1006524209794`.

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009). "The **WEKA** Data Mining Software: An Update." *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18. `doi:10.1145/1656274.1656278`.

Holmes G, Hall M, Frank E (1999). "Generating Rule Sets from Model Trees." In NY Foo (ed.), *Proceedings of the Twelfth Australian Joint Conference on Artificial Intelligence*, pp. 1–12. Springer-Verlag, Heidelberg, Germany.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. `doi:10.1198/106186006x133933`.

Hothorn T, Leisch F, Zeileis A, Hornik K (2005). "The Design and Analysis of Benchmark Experiments." *Journal of Computational and Graphical Statistics*, **14**(3), 675–699. `doi:10.1198/106186005x59630`.

Hothorn T, Zeileis A (2015). "**partykit**: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**, 3905–3909.

Joly A, Schnitzler F, Geurts P, Wehenkel L (2012). "L1-Based Compression of Random Forest Models." In *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Bruges, Belgium.

Kuhn M, Johnson K (2013). *Applied Predictive Modeling*. Springer-Verlag, New York.

Kuhn M, Quinlan JR (2020). ***C50**: C5.0 Decision Trees and Rule-Based Models*. R package version 0.1.3, URL `https://CRAN.R-project.org/package=C50`.

Leisch F, Dimitriadou E (2012). ***mlbench**: Machine Learning Benchmark Problems*. R package version 2.1-1, URL `https://CRAN.R-project.org/package=mlbench`.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL `https://CRAN.R-project.org/doc/Rnews/`.

Lin Y, Jeon Y (2006). "Random Forests and Adaptive Nearest Neighbors." *Journal of the American Statistical Association*, **101**(474), 578–590. `doi:10.1198/016214505000001230`.

Meinshausen N (2010). "Node Harvest." *The Annals of Applied Statistics*, **4**(4), 2049–2072. `doi:10.1214/10-aoas367`.

Nalenz M, Villani M (2018). ***horserule**: Flexible Non-Linear Regression with the HorseRule Algorithm.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=horserule`.

Quinlan JR (1992). "Learning with Continuous Classes." In *5th Australian Joint Conference on Artificial Intelligence*, volume 92, pp. 343–348. World Scientific, Singapore.

Quinlan JR (1993). ***C4.5**: Programs for Machine Learning.* Morgan Kaurfmann, San Mateo.

R Core Team (2019). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.

RuleQuest (2017). *Is **See5/C5.0** Better than **C4.5**?* RuleQuest. URL `https://rulequest.com/see5-comparison.html`.

Schauerhuber M, Zeileis A, Meyer D, Hornik K (2007). "Benchmarking Open-Source Tree Learners in R/**RWeka**." In C Preisach, H Burkhardt, L Schmidt-Thieme, R Decker (eds.), *Data Analysis, Machine Learning and Applications*, pp. 389–396. Springer-Verlag, Heidelberg.

Shimokawa T, Li L, Yan K, Kitamura S, Goto M (2014). "Modified Rule Ensemble Method for Binary Data and Its Applications." *Behaviormetrika*, **41**(2), 225–244. `doi:10.2333/bhmk.41.225`.

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. `doi:10.1037/a0016973`.

Therneau T, Atkinson B, Ripley B (2019). ***rpart**: Recursive Partitioning and Regression Trees.* R package version 4.1-15, URL `https://CRAN.R-project.org/package=rpart`.

Weiss SM, Indurkhya N (2000). "Lightweight Rule Induction." In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1135–1142. Morgan Kaufmann Publishers Inc., San Mateo.

Yang W, Zhang S, Chen Y, Chen Y, Li W, Lu H (2008). "Mining Diagnostic Rules of Breast Tumor on Ultrasound Image Using Cost-Sensitive RuleFit Method." In *International Conference on Intelligent System and Knowledge Engineering (ISKE 2008)*, pp. 354–359. IEEE, Xiamen.

Zeileis A, Hothorn T, Hornik K (2008). "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics*, **17**(2), 492–514. `doi:10.1198/106186008x319331`.

Affiliation:

Marjolein Fokkema
Department of Methods and Statistics
Institute of Psychology
Faculty of Social Sciences
Universiteit Leiden
Leiden, The Netherlands
E-mail: m.fokkema@fsw.leidenuniv.nl

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
February 2020, Volume 92, Issue 12 *Submitted:* 2017-08-05
doi:10.18637/jss.v092.i12 *Accepted:* 2018-10-29