Fitting a prediction rule ensemble

This document provides a manual on how to fit prediction rule ensembles (PREs) as in the example on chronic depression from Fokkema & Strobl (2020). In what follows, the code and results of fitting PREs using **R** package **pre** is provided, intermingled with comments and explanations. Some experience in **R** is recommended (e.g., loading data, fitting a regression model using function `lm()`).

Because we do not own the datasets analyzed in the main paper, in this tutorial we replicate the analyses on an artificially generated dataset. These datas were generated so that the univariate distributions are the same as in the original dataset, but the inter-correlations between variables are different. Therefore, results will differ from those presented in the main paper and do not represent accurate empirical results.

The results in this document have been obtained using **R** version 4.3.1 using packages **pre** version 1.0.7, **partykit** version 1.2-16 and **glmnet** version 4.1-6.

Installing and loading the package

Before we start the analyses, we first install- and load package **pre** by typing in **R**:

```r
install.packages("pre")
library("pre")
```

Predicting Chronic Depression

We replicate the analyses on predicting chronic depression using the `depression.txt` file. After downloading the file, it should be made available in **R**'s current working directory. We can then load the data into **R** by typing:

```r
depression <- read.table("depression.txt", stringsAsFactors = TRUE)
```

We can check the number of columns (variables) and rows (observations) of the dataset by typing:

```r
dim(depression)
```

```
[1] 682  21
```

To get an overview of the types of variables in the dataset, we use function `head` to print the first six rows:

```r
head(depression)
```

```
  dep             disType   Sexe Age edu_yrs IDS BAI FQ LCImax pedigree  alcohol
1  No   comorbid disorder female  43      15  18  14  6  0.093      Yes Positive
2 Yes depressive disorder female  38      15  14  15  0  0.367      Yes Positive
3  No depressive disorder female  26       9  22   6 20  1.000      Yes Positive
4 Yes depressive disorder female  30      11   8   8 11  1.000       No Positive
5 Yes   comorbid disorder female  36      12  13  15  0  0.583      Yes Positive
6  No   comorbid disorder female  40      15  17  27  7  0.912      Yes Positive
          TypeDep  SocPhob      GAD    Panic      Ago AO RemDis
1 First onset MDD Negative Negative Negative Negative 16     No
2   Recurrent MDD Negative Negative Negative Negative 36    Yes
3 First onset MDD Negative Negative Positive Negative 12     No
4   Recurrent MDD Negative Negative Negative Negative 10     No
5   Recurrent MDD Positive Negative Negative Negative 12     No
6   Recurrent MDD Negative Negative Positive Negative 51     No
                    sample ADuse PsychTreat
1             Primary care   Yes        Yes
2 Spec. mental health care   Yes        Yes
3             Primary care   Yes         No
4       General population    No         No
5 Spec. mental health care    No         No
6 Spec. mental health care    No        Yes
```

The first variable in the dataset (`dep`) is the response variable; it is an indicator for whether subjects still meet the criteria of depression two years after baseline (i.e., a chronic depression trajectory). The other variables are potential predictors measured at baseline and are described in more detail in the main paper.

Fitting a PRE requires random sampling of the training observations for generating rules. We therefore first have to set the state of **R**'s random number generator, which will allow for exact replication of the results at a later time:

```r
set.seed(1)
```

We will now fit the ensemble using function `pre()`. The first argument of this function specifies the model to be fitted: the response and potential predictor variables, separated by a tilde (`~`). Here we use the dot (`.`)

as short-hand notation for regressing the specified response (`dep`) on all remaining variables in the dataset. Because the response is a binary factor, we also specify `family = "binomial"`:

```
depression.ens <- pre(dep ~ ., data = depression, family = "binomial",
                      relax = TRUE)
depression.ens
```

```
Final ensemble with cv error within 1se of minimum:

  lambda =  0.04267839
  gamma =  0.25
  number of terms = 3
  mean cv error (se) = 1.30332 (0.01308682)

  cv error type : Binomial Deviance

         rule  coefficient                description
  (Intercept)   -0.1387198                          1
       rule83    0.6469343  LCImax > 0.273 & IDS > 12
       rule74   -0.5614152        IDS <= 16 & AO > 16
       rule53    0.2792436  IDS > 11 & LCImax > 0.265
```

We can obtain a summary of the fitted ensemble as follows:

```
summary(depression.ens)
```

```
Final ensemble with cv error within 1se of minimum:

  lambda =  0.04267839
  gamma =  0.25
  number of terms = 3
  mean cv error (se) = 1.30332 (0.01308682)
```

The results indicate the criterion used for selecting the optimal value of the penalty parameter λ and the mixing parameter γ: the values yielding cross-validated prediction error within one standard error of the minimum. Furthermore, the results indicate that 3 terms were selected in the final PRE. Although the cross-validated error of the selected λ and γ values are reported, it should be noted that this estimate likely provides an overly optimistic value of the expected prediction error, as it was calculated using the same data as was used to generate the rules. Later, we will use function `cvpre()` to obtain a more realistic estimate of future prediction error.

To further inspect the ensemble, we can print it as follows:

```
depression.ens
```

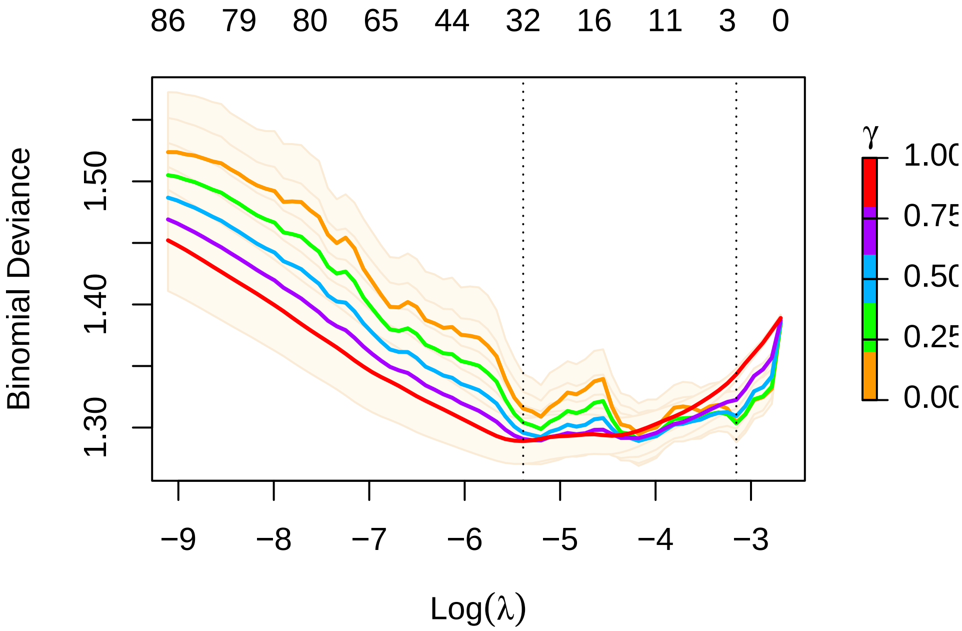
```
         rule  coefficient                 description
  (Intercept)   -0.1387198                           1
       rule83    0.6469343  LCImax > 0.273 & IDS > 12
       rule74   -0.5614152        IDS <= 16 & AO > 16
       rule53    0.2792436  IDS > 11 & LCImax > 0.265
```

Alternatively, we could have typed `print(depression.ens)`, which would have yielded the exact same result. The printed results provide a description of the rules and/or linear terms included in the final ensemble, with their respective coefficients. Note that in this case, no linear terms were selected, as the column `rule` only contains (numbered) rules. If linear terms were selected, this column would also show the names of the selected predictor variables.

The rules are ordered by the absolute value of their coefficients. The coefficient of rule rule83 (LCImax $>$ 0.273 & IDS $>$ 12) indicates that meeting the criteria of this rule increases the log odds of a chronic depression by about 0.65. Note that all rules involve only three variables: `LCImax` (proportion of time in which symptoms of anxiety or depression were present in the four years prior to baseline), `IDS` (psychological test score reflecting severity of depressive symptoms) and `AO` (age of disorder onset) and `Age` (in years).

To inspect the effect of λ and γ, we can request the following plot:

```
plot(depression.ens$glmnet.fit)
```



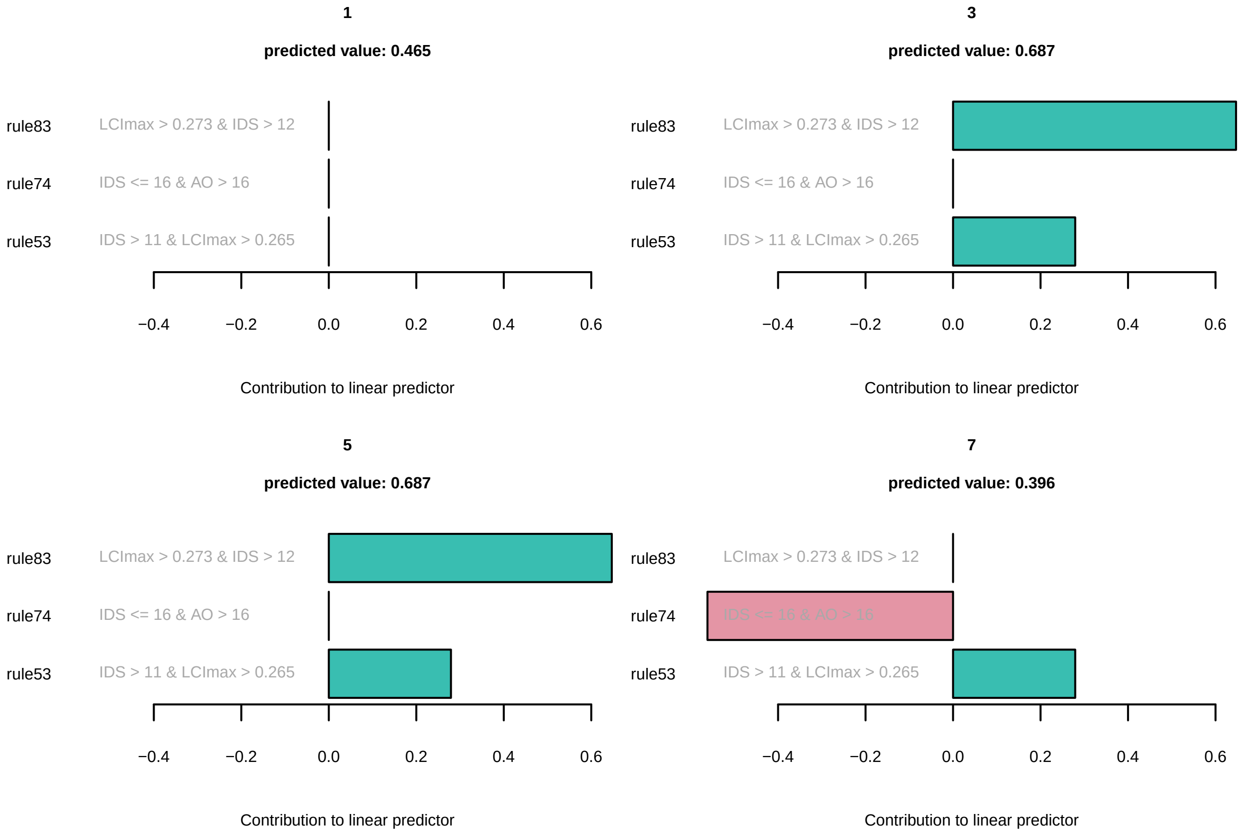
To generate predictions for new observations, the contributions of each rule and linear term in the final ensemble need to be computed and summed. Predictions can be computed using the `predict` method, which requires the user to supply the fitted ensemble and the `newdata` argument, which should supply a dataframe of observations for which predictions will be computed. If the `newdata` argument is not specified, predictions for the original training observations are returned. Here, we request predictions for four of the training observations. By default, the `predict` method returns predictions on the scale of the linear predictors. Through specifying `type = "response"`, we obtain the predicted probabilities:

```
predict(depression.ens, newdata = depression[c(1, 3, 5, 7),], type = "response")
```

```
        1         3         5         7 
0.4653756 0.6872853 0.6872853 0.3963035 
```

Observations 1 and 7 obtained somewhat lower predicted probabilities, while observations 3 and 5 obtained somewhat higher predicted probabilities. We can use function `explain()` to provide a visual explanation of the predictions:

```
expl <- explain(depression.ens, newdata = depression[c(1, 3, 5, 7),])
```



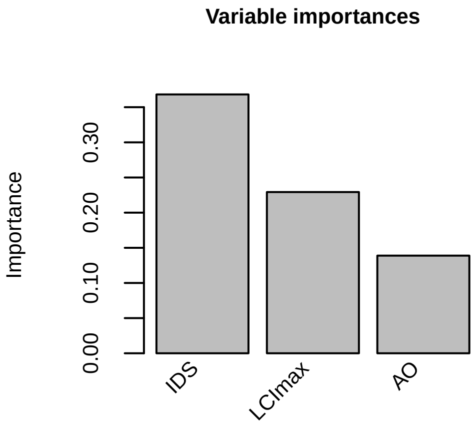
Numerical results are saved in `expl$predictors` (which provides the values of the predictor variables) and `expl$contribution` (which provides the contribution of each term to the individual predictions, as plotted above).

The plotted results show the predicted probabilities, and the contributions of the terms in the final ensemble to the observation-level predictions. Green bars reflect contributions to the predicted value of rules with positive coefficients, while red bars represent contributions of rules with negative coefficients. The absence of a vertical green or red bar indicates that the observation did not meet the conditions of that rule. The rules are ordered from top to bottom in ascending order of global importance. Note that the contributions are on the scale of the linear predictor, thus reflecting the increase in log odds of belonging to the target class (i.e., having a chronic depression trajectory).

The first plot reveals that observation 1 did not meet the conditions of any rule. The predicted probability for this observation is therefore based on the value of the intercept only: $\frac{e^{-0.139}}{1+e^{-0.139}} = 0.465$. Observations 3 and 5 obtained a higher predicted probability, because they meet the conditions of several rules with positive coefficients. Observation 7 meets the conditions of several rules with negative coefficients, resulting in a lower predicted probability.

To obtain an overview of the importances of baselearners (rules and/or linear terms) and predictor variables, we use the `importance()` function, which by default creates a plot of the variable importances:

```
depression.imp <- importance(depression.ens)
```



The plot indicates that the ensemble included only four of the potential predictor variables, which we also observed through inspecting the rules. The remaining variables were not part of any rule or linear term in the final ensemble and thus obtained importances of 0. In addition to plotting the predictor variable importances, function `importance()` invisibly returns a list of variable and baselearner importances, which we have assigned to the `depression.imp` object with the code above. We can access the numeric values of these importances as follows:

```
depression.imp$varimps
```

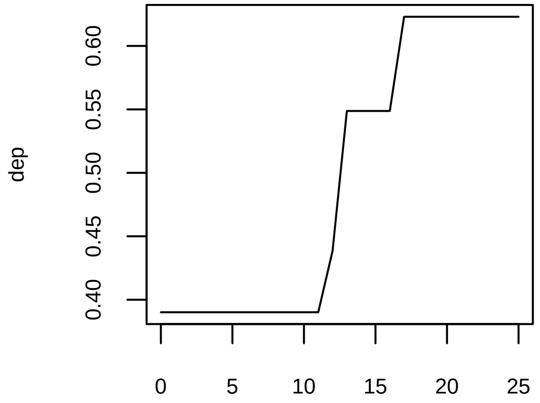
```
  varname       imp
1     IDS 0.3681429
2  LCImax 0.2292659
3      AO 0.1388770
```

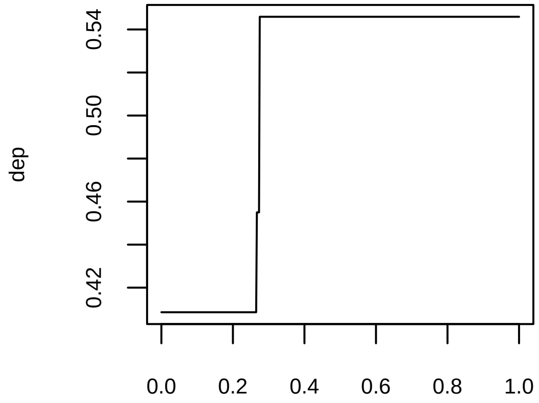
```
depression.imp$baseimps
```

```
    rule                description       imp coefficient        sd
1 rule83 LCImax > 0.273 & IDS > 12 0.3188225   0.6469343 0.4928205
2 rule74       IDS <= 16 & AO > 16 0.2777541  -0.5614152 0.4947392
3 rule53 IDS > 11 & LCImax > 0.265 0.1397093   0.2792436 0.5003132
```

We can obtain univariate partial dependence plots using the `singleplot()` function. We use the `varname` argument to specify the name of the predictor variable for which we want to plot the partial dependence:

```r
singleplot(depression.ens, varname = "IDS")
singleplot(depression.ens, varname = "LCImax")
```

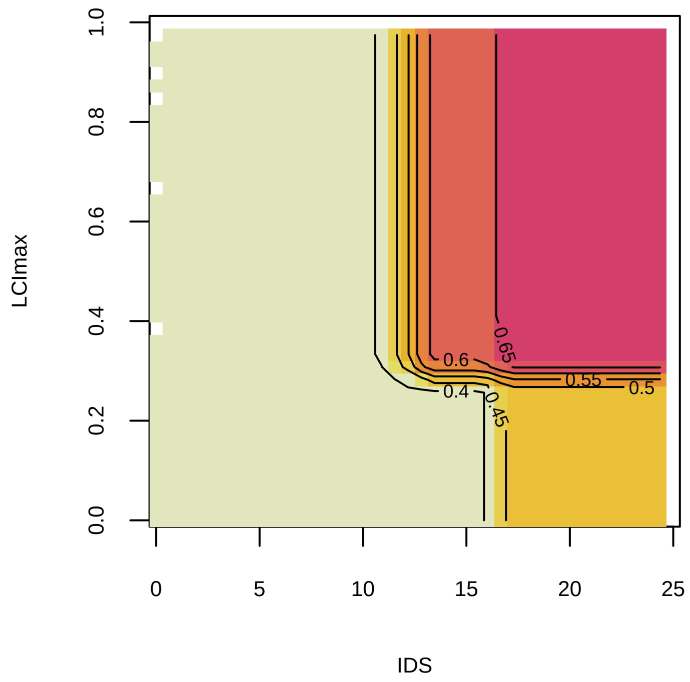




The univariate partial dependence plots reveal a monotonously increasing effect of both `IDS` and `LCImax` on the predicted probability of a chronic trajectory.

We can obtain a bivariate partial dependence plot using the `pairplot()` function. We specify the names of two predictor variables with the `varnames` argument:

```r
pairplot(depression.ens, varnames = c("IDS", "LCImax"))
```



In the bivariate partial dependence plot, the yellow (lighter) areas correspond to lower predicted probabilities, while the red (darker) areas correspond to higher predicted probabilities. The contour lines depict areas with similar predicted values. Like the univariate plots, the bivariate partial dependence plot also reveals that the predicted probability of a chronic trajectory increases with increasing values of `IDS` and/or `LCImax`.

To obtain a realistic estimate of the fitted ensemble's prediction error on future observations, we use function `cvpre()`. This function estimates the expected predictive accuracy using k-fold cross validation. It first separates the original training data into k (approximately) equally sized test samples. Each of the k test samples is then used to assess predictive accuracy of a model fitted on the remaining training observations (i.e., observations that are not in the current test sample). This model is fitted using the same settings as those used for fitting the original model (i.e., `depression.ens`, which was fitted using the default settings). By default, `cvpre()` performs ten-fold cross validation, but a different number of folds can be specified through argument `k`. Random sampling is used to assign observations to folds, so the results depend on the random seed we have set above.

```
set.seed(2)
cv.depression <- cvpre(depression.ens, relax = TRUE)
```

```
$SEL
       SEL         se 
0.23887962 0.00420607 

$AEL
        AEL          se 
0.475875609 0.004270932 

$MCR
[1] 0.3973607

$table
         observed
predicted        No       Yes
      No  0.3328446 0.2360704
      Yes 0.1612903 0.2697947
```

The printed results show the squared error loss (`SEL`) and absolute error loss (`AEL`) with their respective standard errors, the misclassification rate (`MCR`) and a confusion matrix (`table`). These are also stored in `cv.depression$accuracy` for possible later use. The `MCR` indicates that (100-39.74=)60.26 % of observations were correctly classified.

The cross-validated predictions for every observation can be extracted from `cv.depression$cvpreds`. This allows for calculation of alternative accuracy estimates. For example, we may want to calculate the correlation between the predicted probability of belonging to the target class and the observed class membership (a.k.a. the point-biserial correlation):

```
cor(cv.depression$cvpreds, as.numeric(depression$dep))
```

```
          [,1]
[1,] 0.2112699
```

Adjusting and Optimizing Parameters

In the examples above, we have mostly employed default settings of function `pre()`. However, `pre()` has several arguments for controlling the model-fitting procedure. By adjusting these settings, users can fine-tune accuracy and complexity of the final ensemble. The default settings of `pre()` represent the author's choice of 'sensible defaults': settings that are expected to work well out-of-the-box, yielding relatively accurate and sparse ensembles.

However, sometimes users may prefer to use different settings, based on their subject-matter knowledge or specific requirements for application of the results. For example, maximum rule length may be specified based on a researcher's prior knowledge about the order of interactions present in the data, or because rules defined by multiple conditions may be too complex or costly to evaluate in practical applications. Or, a researcher may be more interested in maximizing predictive accuracy than in minimizing complexity, vice versa.

Below, we discuss the parameters that can be adjusted to optimize accuracy, complexity and/or computation time. An extensive explanation of all arguments is provided in the help files, which can be accessed by typing `?`, followed by the function's name. For example, we can access the help files for functions `pre()` and `importance()`, and the `predict` method, as follows:

```
?pre
?importance
?predict.pre
```

Below, we list the most important arguments of function `pre()`, their default values and how they most likely affect complexity, accuracy and/or computation time. We distinguish between 'Model-Fitting' and 'Model-Selection' parameters, where the former control how the initial ensemble of rules and/or linear functions is generated, and the latter control how the final ensemble is selected.

The set of parameter values that will provide optimal predictive accuracy may depend on the data problem at hand. Therefore, in the subsection "Tuning parameters for optimal predictive accuracy", we will provide an example of fine-tuning the parameter values using cross validation, in order to maximize the expected predictive accuracy of the final ensemble.

Model-Fitting Parameters

The following arguments can be passed to function `pre()` and determine how the initial ensemble of rules and/or linear terms is generated:

- `type`: This argument specifies the type of ensemble generated: `"both"` (the default) yields an initial ensemble of rules and linear terms. Alternatively, `"rules"` yields an initial ensemble of rules only and `"linear"` yields an ensemble of linear terms only.

- `ntrees`: Specifies the total number of trees to generate for rule induction. The default (`500`) corresponds to the default value of most random-forest algorithms. Lower values yield lower computation time and likely yield less complex, but also less accurate final ensembles. Higher values likely yield more complex ensembles and may increase the likelihood of overfitting.

- `sampfrac`: Specifies the fraction of randomly selected training observations used for fitting each tree. The default (`.5`) yields subsamples consisting of 50% of the training observations. Values between 0 and 1 yield sampling without replacement (i.e., subsampling). A value of 1 yields sampling with replacement (i.e., bootstrap sampling). Larger values may yield more complex final ensembles and somewhat higher computation times.

- `maxdepth`: Specifies the maximum number of conditions per rule. The default is 3, which yields rules consisting of at most three conditions. A value of 1 yields an additive model, with main effects only. Higher values allow for accommodating (higher-order) interactions, but also increase complexity of the final ensemble and may increase the likelihood of overfitting.

- `tree.unbiased`: Specifies whether unbiased recursive partitioning should be employed for rule generation. The default (`TRUE`) is to employ unbiased recursive partitioning as implemented in package **partykit** (Hothorn & Zeileis, 2015). If set to `FALSE`, the (biased) classification and regression trees algorithm (Breiman et al., 1984) as implemented in package **rpart** (Therneau et al., 2017) will be employed. The latter reduces computation time, but will also yield more complex ensembles and possibly lower predictive accuracy.

- `learnrate`: Specifies the learning rate or boosting parameter applied in sequential tree induction. This parameter specifies the extent to which the response variable is 'corrected' for the predictions of earlier trees, prior to growing a new tree. A value of 0 yields no influence of earlier trees on later trees, while higher values yield increasing influence of earlier trees. Small, non-zero learning rates have been found to perform well in most problems (Friedman & Popescu, 2003), which is reflected in the default value of `.01`. Higher values of the learning rate may yield less complex final ensembles.

- `mtry`: Specifies the number of predictor variables randomly selected as candidates for each split in each tree. The default (`Inf`) takes all potential predictors as candidates for each split. Specifying values > 0 and $< p$ (where p is the number of possible predictor variables) yields a random-forest style approach to rule induction and may decrease correlation between rules in the initial ensemble, which will reduce computation time and may improve predictive accuracy.

- `winsfrac`: Specifies the quantiles of the data distribution to be used for winsorizing (or censoring) linear terms, to reduce the effect of possible outliers. The default is `.05`, resulting in values lower than the .05 and higher than the .95 quantiles of a predictor variable's distribution to be set to the value of the .05 and .95 quantile, respectively. Lower values of `winsfrac` increase the effect of possible outliers. If set to 0, no winsorizing is performed.

Model-Selection Parameter

The final ensemble is selected through penalized regression of the response variable on the rules and linear terms in the initial ensemble. Internally, package **pre** employs package **glmnet** (Friedman et al., 2010) to perform this penalized regression. To obtain the optimal value for the penalty parameter λ, k-fold cross validation is used, with $k = 10$, by default. Parameter λ can take values between 0 and 1, with a value of 0 yielding an unpenalized solution and a value of 1 yielding an intercept-only solution.

Use of the relaxed lasso (Hastie et al., 2020), which can be invoked through specifying `relax = TRUE` in the call to function `pre()`, additionally debiases the lasso-penalized coefficients. With the standard lasso, the higher λ parameter optimal for selection are often suboptimal for prediction. The relaxed lasso adjusts for this.

The *optimal* value of λ is determined, based on one of two possible criteria that can be passed to the `penalty.par.val` argument: `"lambda.min"`, which returns a final ensemble selected with the λ value that yields the minimum cross-validated prediction error. By default, however, `pre()` employs `penalty.par.val = "lambda.1se"`, which returns a final ensemble selected with the λ value that yielded the least complex model, with a cross-validated prediction error within 1 standard error of the minimum. The `"lambda.1se"` criterion generally yields a λ value larger than that of the `"lambda.min"` criterion, in turn yielding less complex final ensembles, that may be less likely to overfit. Although the `"lambda.min'` criterion may yield slightly more accurate final ensembles than the default `lambda.1se` in some cases, it will almost always yield more complex final ensembles. The default criterion `"lambda.1se'` thus favors less complex ensembles, which are less likely to overfit.

The `penalty.par.val` argument can be passed to methods `summary`, `print`, `plot` and `coef`, and functions `importance()`, `singleplot()`, `pairplot()` and `cvpre()`.

References

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

Fokkema, M., & Strobl, C. (2020). Fitting prediction rule ensembles to psychological research data: An introduction and tutorial. *Psychological Methods*, *25*(5), 636.

Friedman, J. H., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22. http://www.jstatsoft.org/v33/i01/

Friedman, J. H., & Popescu, B. E. (2003). *Importance sampled learning ensembles* [Technical Report]. Stanford University.

Hastie, T., Tibshirani, R., & Tibshirani, R. (2020). Best subset, forward stepwise or lasso? Analysis and recommendations based on extensive comparisons. *Statistical Science*, *35*(4), 579–592.

Hothorn, T., & Zeileis, A. (2015). partykit: A modular toolkit for recursive partytioning in R. *Journal of Machine Learning Research*, *16*, 3905–3909.

Therneau, T., Atkinson, B., & Ripley, B. (2017). *Rpart: Recursive partitioning and regression trees*. https://CRAN.R-project.org/package=rpart