Package 'Rforestry'

March 25, 2023

Type Package

Title Random Forests, Linear Trees, and Gradient Boosting for Inference and Interpretability

Version 0.10.0

Maintainer Theo Saarinen <theo_s@berkeley.edu>

BugReports https://github.com/forestry-labs/Rforestry/issues

URL https://github.com/forestry-labs/Rforestry

Description Provides fast implementations of Honest Random Forests, Gradient Boosting, and Linear Random Forests, with an emphasis on inference and interpretability. Additionally contains methods for variable importance, out-of-bag prediction, regression monotonicity, and several methods for missing data imputation. Soren R. Kunzel, Theo F. Saarinen, Edward W. Liu, Jasjeet S. Sekhon (2019) <arXiv:1906.06463>.

License GPL (>= 3)

Encoding UTF-8

Imports Rcpp (>= 0.12.9), parallel, methods, visNetwork, glmnet (>= 4.1), grDevices, onehot, pROC

LinkingTo Rcpp, RcppArmadillo, RcppThread

RoxygenNote 7.2.3

Suggests testthat, knitr, rmarkdown, mvtnorm

Collate 'R_preprocessing.R' 'RcppExports.R' 'forestry.R' 'backwards_compatible.R' 'compute_rf_lp.R' 'neighborhood_imputation.R' 'plottree.R'

NeedsCompilation yes

Author Sören Künzel [aut], Theo Saarinen [aut, cre], Simon Walter [aut], Sam Antonyan [aut], Edward Liu [aut], Allen Tang [aut], Jasjeet Sekhon [aut]

Repository CRAN

Date/Publication 2023-03-25 00:50:02 UTC

R topics documented:

`addTrees` *addTrees-forestry*

Description

Add more trees to the existing forest.

Usage

```
addTrees(object, ntree)
```

Arguments

`object`	A ‘forestry‘ object.
`ntree`	Number of new trees to add

Value

A ‘forestry‘ object

`compute_lp-forestry` *compute lp distances*

Description

Return the L_p norm distances of selected test observations relative to the training observations which the forest was trained on.

Usage

```
compute_lp(
  object,
  newdata,
  feature,
  p,
  scale = FALSE,
  aggregation = "average",
  trainingIdx = NULL
)
```

Arguments

`object`	A ‘forestry‘ object.
`newdata`	A data frame of test predictors.
`feature`	A string denoting the dimension for computing lp distances.
`p`	A positive real number determining the norm p-norm used.
`scale`	A boolean indicating whether or not we want to center + scale the features (based on the mean and sd of the training data) before calculating the L_p norm. This is useful for computing the detachment index, but can be less useful when we need to interpret the L_p distances.
`aggregation`	The aggregation used when the weightMatrix is calculated. This can be useful for calculating the lp distances on observations in the training data. This must be one of ‘average‘, ‘oob‘, or ‘doubleOOB‘. When newdata has fewer rows than the training data, one must also pass the vector of training indices corresponding to the indices of the observations in the original data set. Default is ‘average‘.
`trainingIdx`	This is an optional parameter that must be set when aggregation is set to ‘oob‘ or ‘doubleOOB‘ and the newdata is not the same size as the training data.

Value

A vector of the lp distances.

Examples

```
# Set seed for reproductivity
set.seed(292313)

# Use Iris Data
test_idx <- sample(nrow(iris), 11)
x_train <- iris[-test_idx, -1]
y_train <- iris[-test_idx, 1]
x_test <- iris[test_idx, -1]

rf <- forestry(x = x_train, y = y_train,nthread = 2)
predict(rf, x_test)

# Compute the l2 distances in the "Petal.Length" dimension
distances_2 <- compute_lp(object = rf,
                          newdata = x_test,
                          feature = "Petal.Length",
                          p = 2)
```

`CppToR_translator` *Cpp to R translator*

Description

Add more trees to the existing forest.

Usage

```
CppToR_translator(object)
```

Arguments

`object`	external CPP pointer that should be translated from Cpp to an R object

Value

A list of lists. Each sublist contains the information to span a tree.

forestry *forestry*

Description

forestry

Usage

```
forestry(
  x,
  y,
  ntree = 500,
  replace = TRUE,
  sampsize = if (replace) nrow(x) else ceiling(0.632 * nrow(x)),
  sample.fraction = NULL,
  mtry = max(floor(ncol(x)/3), 1),
  nodesizeSpl = 5,
  nodesizeAvg = 5,
  nodesizeStrictSpl = 1,
  nodesizeStrictAvg = 1,
  minSplitGain = 0,
  maxDepth = round(nrow(x)/2) + 1,
  interactionDepth = maxDepth,
  interactionVariables = numeric(0),
  featureWeights = NULL,
  deepFeatureWeights = NULL,
  observationWeights = NULL,
  customSplitSample = NULL,
  customAvgSample = NULL,
  customExcludeSample = NULL,
  splitratio = 1,
  OOBhonest = FALSE,
  doubleBootstrap = if (OOBhonest) TRUE else FALSE,
  seed = as.integer(runif(1) * 1000),
  verbose = FALSE,
  nthread = 0,
  splitrule = "variance",
  middleSplit = FALSE,
  maxObs = length(y),
  linear = FALSE,
  linFeats = 0:(ncol(x) - 1),
  monotonicConstraints = rep(0, ncol(x)),
  groups = NULL,
  minTreesPerFold = 0,
  foldSize = 1,
  monotoneAvg = FALSE,
```

```
  overfitPenalty = 1,
  scale = TRUE,
  doubleTree = FALSE,
  naDirection = FALSE,
  reuseforestry = NULL,
  savable = TRUE,
  saveable = TRUE
)
```

Arguments

x	A data frame of all training predictors.
y	A vector of all training responses.
ntree	The number of trees to grow in the forest. The default value is 500.
replace	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
sampsize	The size of total samples to draw for the training data. If sampling with replacement, the default value is the length of the training data. If sampling without replacement, the default value is two-thirds of the length of the training data.
sample.fraction	If this is given, then sampsize is ignored and set to be round(length(y) * sample.fraction). It must be a real number between 0 and 1
mtry	The number of variables randomly selected at each split point. The default value is set to be one-third of the total number of features of the training data.
nodesizeSpl	Minimum observations contained in terminal nodes. The default value is 5.
nodesizeAvg	Minimum size of terminal nodes for averaging dataset. The default value is 5.
nodesizeStrictSpl	Minimum observations to follow strictly in terminal nodes. The default value is 1.
nodesizeStrictAvg	The minimum size of terminal nodes for averaging data set to follow when predicting. No splits are allowed that result in nodes with observations less than this parameter. This parameter enforces overlap of the averaging data set with the splitting set when training. When using honesty, splits that leave less than nodesizeStrictAvg averaging observations in either child node will be rejected, ensuring every leaf node also has at least nodesizeStrictAvg averaging observations. The default value is 1.
minSplitGain	Minimum loss reduction to split a node further in a tree.
maxDepth	Maximum depth of a tree. The default value is 99.
interactionDepth	All splits at or above interaction depth must be on variables that are not weighting variables (as provided by the interactionVariables argument).
interactionVariables	Indices of weighting variables.

`featureWeights` (optional) vector of sampling probabilities/weights for each feature used when subsampling mtry features at each node above or at interactionDepth. The default is to use uniform probabilities.

`deepFeatureWeights`
Used in place of featureWeights for splits below interactionDepth.

`observationWeights`
Denotes the weights for each training observation that determine how likely the observation is to be selected in each bootstrap sample. This option is not allowed when sampling is done without replacement.

`customSplitSample`
List of vectors for user-defined splitting observations per tree. The vector at index i contains the indices of the sampled splitting observations, with replacement allowed, for tree i. This feature overrides other sampling parameters and must be set in conjunction with customAvgSample.

`customAvgSample`
List of vectors for user-defined averaging observations per tree. The vector at index i contains the indices of the sampled splitting observations, with replacement allowed, for tree i. This feature overrides other sampling parameters and must be set in conjunction with customSplitSample.

`customExcludeSample`
An optional list of vectors for user-defined excluded observations per tree. The vector at index i contains the indices of the excluded observations for tree i. An observation is considered excluded if it does not appear in the splitting or averaging set and has been explicitly withheld from being sampled for a tree. Excluded observations are not considered out-of-bag, so when we call predict with aggregation = "oob", when we predict for an observation, we will only use the predictions of trees in which the observation was in the customSplitSample (and neither in the customAvgSample nor the customExcludeSample). This parameter is optional even when customSplitSample and customAvgSample are set. It is also optional at the tree level, so can have fewer than ntree entries. When given fewer than ntree entries, for example K, the entries will be applied to the first K trees in the forest and the remaining trees will have no excludedSamples.

`splitratio` Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1 (the default), then the splitting set uses the entire data, as does the averaging set—i.e., the standard Breiman RF setup. If the ratio is 0, then the splitting data set is empty, and the entire dataset is used for the averaging set (This is not a good usage, however, since there will be no data available for splitting).

`OOBhonest` In this version of honesty, the out-of-bag observations for each tree are used as the honest (averaging) set. This setting also changes how predictions are constructed. When predicting for observations that are out-of-sample (using predict(..., aggregation = "average")), all the trees in the forest are used to construct predictions. When predicting for an observation that was in-sample (using predict(..., aggregation = "oob")), only the trees for which that observation was not in the averaging set are used to construct the prediction for that observation. aggregation="oob" (out-of-bag) ensures that the outcome value for an observation is never used to construct predictions for a given observation even when it is

in sample. This property does not hold in standard honesty, which relies on an asymptotic subsampling argument. By default, when OOBhonest = TRUE, the out-of-bag observations for each tree are resamples with replacement to be used for the honest (averaging) set. This results in a third set of observations that are left out of both the splitting and averaging set, we call these the double out-of-bag (doubleOOB) observations. In order to get the predictions of only the trees in which each observation fell into this doubleOOB set, one can run predict(... , aggregation = "doubleOOB"). In order to not do this second bootstrap sample, the doubleBootstrap flag can be set to FALSE.

`doubleBootstrap`
: The doubleBootstrap flag provides the option to resample with replacement from the out-of-bag observations set for each tree to construct the averaging set when using OOBhonest. If this is FALSE, the out-of-bag observations are used as the averaging set. By default this option is TRUE when running OOBhonest = TRUE. This option increases diversity across trees.

`seed`
: random seed

`verbose`
: Indicator to train the forest in verbose mode

`nthread`
: Number of threads to train and predict the forest. The default number is 0 which represents using all cores.

`splitrule`
: Only variance is implemented at this point and it specifies the loss function according to which the splits of random forest should be made.

`middleSplit`
: Indicator of whether the split value is takes the average of two feature values. If FALSE, it will take a point based on a uniform distribution between two feature values. (Default = FALSE)

`maxObs`
: The max number of observations to split on.

`linear`
: Indicator that enables Ridge penalized splits and linear aggregation functions in the leaf nodes. This is recommended for data with linear outcomes. For implementation details, see: https://arxiv.org/abs/1906.06463. Default is FALSE.

`linFeats`
: A vector containing the indices of which features to split linearly on when using linear penalized splits (defaults to use all numerical features).

`monotonicConstraints`
: Specifies monotonic relationships between the continuous features and the outcome. Supplied as a vector of length p with entries in 1,0,-1 which 1 indicating an increasing monotonic relationship, -1 indicating a decreasing monotonic relationship, and 0 indicating no constraint. Constraints supplied for categorical variable will be ignored.

`groups`
: A vector of factors specifying the group membership of each training observation. these groups are used in the aggregation when doing out of bag predictions in order to predict with only trees where the entire group was not used for aggregation. This allows the user to specify custom subgroups which will be used to create predictions which do not use any data from a common group to make predictions for any observation in the group. This can be used to create general custom resampling schemes, and provide predictions consistent with the Out-of-Group set.

`minTreesPerFold`
: The number of trees which we make sure have been created leaving out each fold (each fold is a set of randomly selected groups). This is 0 by default, so we will not give any special treatment to the groups when sampling observations, however if this is set to a positive integer, we modify the bootstrap sampling scheme to ensure that exactly that many trees have each group left out. We do this by, for each fold, creating minTreesPerFold trees which are built on observations sampled from the set of training observations which are not in a group in the current fold. The folds form a random partition of all of the possible groups, each of size foldSize. This means we create at least # folds * minTreesPerFold trees for the forest. If ntree > # folds * minTreesPerFold, we create max(# folds * minTreesPerFold, ntree) total trees, in which at least minTreesPerFold are created leaving out each fold.

`foldSize`
: The number of groups that are selected randomly for each fold to be left out when using minTreesPerFold. When minTreesPerFold is set and foldSize is set, all possible groups will be partitioned into folds, each containing foldSize unique groups (if foldSize doesn't evenly divide the number of groups, a single fold will be smaller, as it will contain the remaining groups). Then minTreesPerFold are grown with each entire fold of groups left out.

`monotoneAvg`
: This is a boolean flag that indicates whether or not monotonic constraints should be enforced on the averaging set in addition to the splitting set. This flag is meaningless unless both honesty and monotonic constraints are in use. The default is FALSE.

`overfitPenalty`
: Value to determine how much to penalize the magnitude of coefficients in ridge regression when using linear splits.

`scale`
: A parameter which indicates whether or not we want to scale and center the covariates and outcome before doing the regression. This can help with stability, so by default is TRUE.

`doubleTree`
: if the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. (Default = FALSE)

`naDirection`
: Sets a default direction for missing values in each split node during training. It test placing all missing values to the left and right, then selects the direction that minimizes loss. If no missing values exist, then a default direction is randomly selected in proportion to the distribution of observations on the left and right. (Default = FALSE)

`reuseforestry`
: Pass in an 'forestry' object which will recycle the dataframe the old object created. It will save some space working on the same data set.

`savable`
: If TRUE, then RF is created in such a way that it can be saved and loaded using save(...) and load(...). However, setting it to TRUE (default) will take longer and use more memory. When training many RF, it makes sense to set this to FALSE to save time and memory.

`saveable`
: deprecated. Do not use.

Value

A 'forestry' object.

Note

Treatment of Missing Data

In version 0.9.0.34, we have modified the handling of missing data. Instead of the greedy approach used in previous iterations, we now test any potential split by putting all NA's to the right, and all NA's to the left, and taking the choice which gives the best MSE for the split. Under this version of handling the potential splits, we will still respect monotonic constraints. So if we put all NA's to either side, and the resulting leaf nodes have means which violate the monotone constraints, the split will be rejected.

Examples

```
set.seed(292315)
library(Rforestry)
test_idx <- sample(nrow(iris), 3)
x_train <- iris[-test_idx, -1]
y_train <- iris[-test_idx, 1]
x_test <- iris[test_idx, -1]

rf <- forestry(x = x_train, y = y_train, nthread = 2)
predict(rf, x_test)

set.seed(49)
library(Rforestry)

n <- c(100)
a <- rnorm(n)
b <- rnorm(n)
c <- rnorm(n)
y <- 4*a + 5.5*b - .78*c
x <- data.frame(a,b,c)

forest <- forestry(
          x,
          y,
          ntree = 10,
          replace = TRUE,
          nodesizeStrictSpl = 5,
          nodesizeStrictAvg = 5,
          nthread = 2,
          linear = TRUE
          )

predict(forest, x)
```

`forestry-class` *forestry class*

Description

'honestRF' class only exists for backwards compatibility reasons

---

`forest_checker` *Checks if forestry object has valid pointer for C++ object.*

Description

Checks if forestry object has valid pointer for C++ object.

Usage

```
forest_checker(object)
```

Arguments

`object`	a forestry object

Value

A message if the forest does not have a valid C++ pointer.

`getOOB-forestry` *getOOB-forestry*

Description

Calculate the out-of-bag error of a given forest. This is done by using the out-of-bag predictions for each observation, and calculating the MSE over the entire forest.

Usage

```
getOOB(object, noWarning)
```

Arguments

`object`	A 'forestry' object.
`noWarning`	flag to not display warnings

Value

The OOB error of the forest.

getOOBpreds-forestry *getOOBpreds-forestry*

Description

Calculate the out-of-bag predictions of a given forest.

Usage

```
getOOBpreds(object, newdata = NULL, doubleOOB = FALSE, noWarning = FALSE)
```

Arguments

object	A trained model object of class "forestry".
newdata	A possible new data frame on which to run out of bag predictions. If this is not NULL, we assume that the indices of newdata are the same as the indices of the training set, and will use these to find which trees the observation is considered in/out of bag for.
doubleOOB	A flag specifying whether or not we should use the double OOB set for the OOB predictions. This is the set of observations for each tree which were in neither the averaging set nor the splitting set. Note that the forest must have been trained with doubleBootstrap = TRUE for this to be used. Default is FALSE.
noWarning	Flag to not display warnings.

Value

The vector of all training observations, with their out of bag predictions. Note each observation is out of bag for different trees, and so the predictions will be more or less stable based on the observation. Some observations may not be out of bag for any trees, and here the predictions are returned as NA.

See Also

forestry

getVI *getVI-forestry*

Description

Calculate the percentage increase in OOB error of the forest when each feature is shuffled.

Usage

```
getVI(object, noWarning, metric = "mse", seed = 1)
```

Arguments

object	A ‘forestry‘ object.
noWarning	flag to not display warnings
metric	A parameter to determine how the predictions of the forest with a permuted variable are compared to the predictions of the standard forest. Must be one of c("mse","auc","tnr"), "mse" gives the percentage increase in mse when the feature is permuted, "auc" gives the percentage decrease in AUC when the feature is permuted, and "tnr" gives the percentage decrease in TNR when the TPR is 99% when the feature is permuted.
seed	A parameter to seed the random number generator for shuffling the features of X.

Value

The variable importance of the forest.

Note

Pass a seed to this function so it is possible to replicate the vector permutations used when measuring feature importance.

---

honestRF *Honest Random Forest*

Description

This function is deprecated and only exists for backwards backwards compatibility. The function you want to use is ‘forestry‘.

Usage

```
honestRF(...)
```

Arguments

...	parameters which are passed directly to ‘forestry‘

Value

A ‘forestry‘ object

impute_features *Feature imputation using random forests neighborhoods*

Description

This function uses the neighborhoods implied by a random forest to impute missing features. The neighbors of a data point are all the training points assigned to the same leaf in at least one tree in the forest. The weight of each neighbor is the fraction of trees in the forest for which it was assigned to the same leaf. We impute a missing feature for a point by computing the weighted average feature value, using neighborhood weights, using all of the point's neighbors.

Usage

```
impute_features(
  object,
  newdata,
  seed = round(runif(1) * 10000),
  use_mean_imputation_fallback = FALSE
)
```

Arguments

object	an object of class 'forestry'
newdata	the feature data.frame we will impute missing features for.
seed	a random seed passed to the predict method of forestry
use_mean_imputation_fallback	if TRUE, mean imputation (for numeric variables) and mode imputation (for factor variables) is used for missing features for which all neighbors also had the corresponding feature missing; if FALSE these missing features remain NAs in the data frame returned by 'impute_features'.

Value

A data.frame that is newdata with imputed missing values.

Examples

```
iris_with_missing <- iris
idx_miss_factor <- sample(nrow(iris), 25, replace = TRUE)
iris_with_missing[idx_miss_factor, 5] <- NA
idx_miss_numeric <- sample(nrow(iris), 25, replace = TRUE)
iris_with_missing[idx_miss_numeric, 3] <- NA

x <- iris_with_missing[,-1]
y <- iris_with_missing[, 1]

forest <- forestry(x, y, ntree = 500, seed = 2,nthread = 2)
imputed_x <- impute_features(forest, x, seed = 2)
```

loadForestry *load RF*

Description

This wrapper function checks the forestry object, makes it saveable if needed, and then saves it.

Usage

```
loadForestry(filename)
```

Arguments

filename	a filename in which to store the 'forestry' object

Value

The loaded forest from filename.

make_savable *make_savable*

Description

When a 'foresty' object is saved and then reloaded the Cpp pointers for the data set and the Cpp forest have to be reconstructed

Usage

```
make_savable(object)
```

Arguments

object	an object of class 'forestry'

Value

A list of lists. Each sublist contains the information to span a tree.

Note

'make_savable' does not translate all of the private member variables of the C++ forestry object so when the forest is reconstructed with 'relinkCPP_prt' some attributes are lost. For example, 'nthreads' will be reset to zero. This makes it impossible to disable threading when predicting for forests loaded from disk.

Examples

```
set.seed(323652639)
x <- iris[, -1]
y <- iris[, 1]
forest <- forestry(x, y, ntree = 3, nthread = 2)
y_pred_before <- predict(forest, x)

forest <- make_savable(forest)

wd <- tempdir()
saveForestry(forest, filename = file.path(wd, "forest.Rda"))
rm(forest)

forest <- loadForestry(file.path(wd, "forest.Rda"))

y_pred_after <- predict(forest, x)

file.remove(file.path(wd, "forest.Rda"))
```

---

plot-forestry *visualize a tree*

Description

plots a tree in the forest.

Usage

```
## S3 method for class 'forestry'
plot(x, tree.id = 1, print.meta_dta = FALSE, beta.char.len = 30, ...)
```

Arguments

x	A forestry x.
tree.id	Specifies the tree number that should be visualized.
print.meta_dta	A flag indicating whether the data for the plot should be printed.
beta.char.len	The length of the beta values in leaf node representation. This is only required when plotting a forestry object with linear aggregation functions (linear = TRUE).
...	additional arguments that are not used.

Details

plot

Value

A plot of the specified tree in the forest.

Examples

```
set.seed(292315)
rf <- forestry(x = iris[,-1],
               y = iris[, 1],
               nthread = 2)

plot(x = rf)
plot(x = rf, tree.id = 2)
plot(x = rf, tree.id = 500)
```

---

predict-forestry *predict-forestry*

Description

Return the prediction from the forest.

Usage

```
## S3 method for class 'forestry'
predict(
  object,
  newdata = NULL,
  aggregation = "average",
  holdOutIdx = NULL,
  trainingIdx = NULL,
  seed = as.integer(runif(1) * 10000),
  nthread = 0,
  exact = NULL,
  trees = NULL,
  weightMatrix = FALSE,
  ...
)
```

Arguments

object	A ‘forestry‘ object.
newdata	A data frame of testing predictors.
aggregation	How the individual tree predictions are aggregated: ‘average‘ returns the mean of all trees in the forest; ‘terminalNodes‘ also returns the weightMatrix, as well as "terminalNodes", a matrix where the ith entry of the jth column is the index of the leaf node to which the ith observation is assigned in the jth tree; and "sparse", a matrix where the ith entry in the jth column is 1 if the ith observation in newdata is assigned to the jth leaf and 0 otherwise. In each tree the leaves are indexed using a depth first ordering, and, in the "sparse" representation, the first

leaf in the second tree has column index one more than the number of leaves in the first tree and so on. So, for example, if the first tree has 5 leaves, the sixth column of the "sparse" matrix corresponds to the first leaf in the second tree. ‘oob‘ returns the out-of-bag predictions for the forest. We assume that the ordering of the observations in newdata have not changed from training. If the ordering has changed, we will get the wrong OOB indices. ‘doubleOOB‘ is an experimental flag, which can only be used when OOBhonest = TRUE and doubleBootstrap = TRUE. When both of these settings are on, the splitting set is selected as a bootstrap sample of observations and the averaging set is selected as a bootstrap sample of the observations which were left out of bag during the splitting set selection. This leaves a third set which is the observations which were not selected in either bootstrap sample. This predict flag gives the predictions using- for each observation- only the trees in which the observation fell into this third set (so was neither a splitting nor averaging example). ‘coefs‘ is an aggregation option which works only when linear aggregation functions have been used. This returns the linear coefficients for each linear feature which were used in the leaf node regression of each predicted point.

`holdOutIdx` This is an optional argument, containing a vector of indices from the training data set that should be not be allowed to influence the predictions of the forest. When a vector of indices of training observations are given, the predictions will be made only with trees in the forest that do not contain any of these indices in either the splitting or averaging sets. This cannot be used at the same time as any other aggregation options. If ‘weightMatrix = TRUE‘, this will return the weightMatrix corresponding to the predictions made with trees respecting holdOutIdx. If there are no trees that have held out all of the indices in holdOutIdx, then the predictions will return NaN.

`trainingIdx` This is an optional parameter to give the indices of the observations in ‘newdata‘ from the training data set. This is used when we want to run predict on only a subset of observations from the training data set and use ‘aggregation = "oob"‘ or ‘aggregation = "doubleOOB"‘. For example, at the tree level, a tree make out of bag (‘aggregation = "oob"‘) predictions for the indices in the set setdiff(trainingIdx,tree$averagingIndices) and will make double out-of-bag predictions for the indices in the set setdiff(trainingIdx,union(tree$averagingIndices,tree$splittingIndices). Note, this parameter must be set when predict is called with an out-of-bag aggregation option on a data set not matching the original training data size. The order of indices in ‘trainingIdx‘ also needs to match the order of observations in newdata. So for an arbitrary index set ‘trainingIdx‘ and dataframe ‘newdata‘, of the same size as the training set, the predictions from ‘predict(rf, newdata[trainingIdx,],‘ ‘aggregation = "oob", trainingIdx = trainingIdx)‘ should match the predictions of to ‘predict(rf, newdata, aggregation = "oob")[trainingIdx]‘. This option also works with the ‘weightMatrix‘ option and will return the (smaller) weightMatrix for the observations in the passed data frame.

`seed` random seed

`nthread` The number of threads with which to run the predictions with. This will default to the number of threads with which the forest was trained with.

`exact` This specifies whether the forest predictions should be aggregated in a reproducible ordering. Due to the non-associativity of floating point addition, when

we predict in parallel, predictions will be aggregated in varied orders as different threads finish at different times. By default, exact is TRUE unless N > 100,000 or a custom aggregation function is used.

`trees` A vector (1-indexed) of indices in the range 1:ntree which tells predict which trees in the forest to use for the prediction. Predict will by default take the average of all trees in the forest, although this flag can be used to get single tree predictions, or averages of diffferent trees with different weightings. Duplicate entries are allowed, so if trees = c(1,2,2) this will predict the weighted average prediction of only trees 1 and 2 weighted by: predict(..., trees = c(1,2,2)) = (predict(..., trees = c(1)) + 2*predict(..., trees = c(2))) / 3. note we must have exact = TRUE, and aggregation = "average" to use tree indices.

`weightMatrix` An indicator of whether or not we should also return a matrix of the weights given to each training observation when making each prediction. When getting the weight matrix, aggregation must be one of ‘average‘, ‘oob‘, and ‘doubleOOB‘.

`...` additional arguments.

Value

A vector of predicted responses.

`predictInfo` *predictInfo-forestry*

Description

Get the observations which are used to predict for a set of new observations using either all trees (for out of sample observations), or tree for which the observation is out of averaging set or out of sample entirely.

Usage

```
predictInfo(object, newdata, aggregation = "average")
```

Arguments

`object` A ‘forestry‘ object.

`newdata` Data on which we want to do predictions. Must be the same length as the training set if we are doing ‘oob‘ or ‘doubleOOB‘ aggregation.

`aggregation` Specifies which aggregation version is used to predict for the observation, must be one of ‘average‘,‘oob‘, and ‘doubleOOB‘.

Value

A list with four entries. ‘weightMatrix‘ is a matrix specifying the weight given to training observatio i when prediction on observation j. ‘avgIndices‘ gives the indices which are in the averaging set for each new observation. ‘avgWeights‘ gives the weights corresponding to each averaging observation returned in ‘avgIndices‘. ‘obsInfo‘ gives the full observation vectors which were used to predict for an observation, as well as the weight given each observation.

`preprocess_testing` *preprocess_testing*

Description

Perform preprocessing for the testing data, including converting data to dataframe, and testing if the columns are consistent with the training data and encoding categorical data into numerical representation in the same way as training data.

Usage

```
preprocess_testing(x, categoricalFeatureCols, categoricalFeatureMapping)
```

Arguments

`x`	A data frame of all training predictors.
`categoricalFeatureCols`	A list of index for all categorical data. Used for trees to detect categorical columns.
`categoricalFeatureMapping`	A list of encoding details for each categorical column, including all unique factor values and their corresponding numeric representation.

Value

A preprocessed training dataaset x

`preprocess_training` *preprocess_training*

Description

Perform preprocessing for the training data, including converting data to dataframe, and encoding categorical data into numerical representation.

Usage

```
preprocess_training(x, y)
```

Arguments

x	A data frame of all training predictors.
y	A vector of all training responses.

Value

A list of two datasets along with necessary information that encodes the preprocessing.

---

relinkCPP_prt *relink CPP ptr*

Description

When a 'foresty' object is saved and then reloaded the Cpp pointers for the data set and the Cpp forest have to be reconstructed

Usage

```
relinkCPP_prt(object)
```

Arguments

object	an object of class 'forestry'

Value

Relinks the pointer to the correct C++ object.

---

saveForestry *save RF*

Description

This wrapper function checks the forestry object, makes it saveable if needed, and then saves it.

Usage

```
saveForestry(object, filename, ...)
```

Arguments

object	an object of class 'forestry'
filename	a filename in which to store the 'forestry' object
...	additional arguments useful for specifying compression type and level

Value

Saves the forest into filename.

scale_center *scale_center*

Description

Given a dataframe, scale and center the continous features

Usage

```
scale_center(x, categoricalFeatureCols, colMeans, colSd)
```

Arguments

x	A dataframe in order to be processed.
categoricalFeatureCols	A vector of the categorical features, we don't want to scale/center these. Should be 1-indexed.
colMeans	A vector of the means to center each column.
colSd	A vector of the standard deviations to scale each column with.

Value

A scaled and centered dataset x

testing_data_checker-forestry *Test data check*

Description

Check the testing data to do prediction

Usage

```
testing_data_checker(object, newdata, hasNas)
```

Arguments

object	A forestry object.
newdata	A data frame of testing predictors.
hasNas	TRUE if the there were NAs in the training data FALSE otherwise.

Value

A feature dataframe if it can be used for new predictions.

`training_data_checker` *Training data check*

Description

Check the input to forestry constructor

Usage

```
training_data_checker(
  x,
  y,
  ntree,
  replace,
  sampsize,
  mtry,
  nodesizeSpl,
  nodesizeAvg,
  nodesizeStrictSpl,
  nodesizeStrictAvg,
  minSplitGain,
  maxDepth,
  interactionDepth,
  splitratio,
  OOBhonest,
  doubleBootstrap,
  nthread,
  middleSplit,
  doubleTree,
  linFeats,
  monotonicConstraints,
  groups,
  featureWeights,
  deepFeatureWeights,
  observationWeights,
  customSplitSample,
  customAvgSample,
  customExcludeSample,
  linear,
  scale,
  hasNas,
  naDirection
)
```

Arguments

x	A data frame of all training predictors.

`y` A vector of all training responses.

`ntree` The number of trees to grow in the forest. The default value is 500.

`replace` An indicator of whether sampling of training data is with replacement. The default value is TRUE.

`sampsize` The size of total samples to draw for the training data. If sampling with replacement, the default value is the length of the training data. If sampling without replacement, the default value is two-thirds of the length of the training data.

`mtry` The number of variables randomly selected at each split point. The default value is set to be one-third of the total number of features of the training data.

`nodesizeSpl` Minimum observations contained in terminal nodes. The default value is 5.

`nodesizeAvg` Minimum size of terminal nodes for averaging dataset. The default value is 5.

`nodesizeStrictSpl`
Minimum observations to follow strictly in terminal nodes. The default value is 1.

`nodesizeStrictAvg`
The minimum size of terminal nodes for averaging data set to follow when predicting. No splits are allowed that result in nodes with observations less than this parameter. This parameter enforces overlap of the averaging data set with the splitting set when training. When using honesty, splits that leave less than nodesizeStrictAvg averaging observations in either child node will be rejected, ensuring every leaf node also has at least nodesizeStrictAvg averaging observations. The default value is 1.

`minSplitGain` Minimum loss reduction to split a node further in a tree.

`maxDepth` Maximum depth of a tree. The default value is 99.

`interactionDepth`
All splits at or above interaction depth must be on variables that are not weighting variables (as provided by the interactionVariables argument).

`splitratio` Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1 (the default), then the splitting set uses the entire data, as does the averaging set—i.e., the standard Breiman RF setup. If the ratio is 0, then the splitting data set is empty, and the entire dataset is used for the averaging set (This is not a good usage, however, since there will be no data available for splitting).

`OOBhonest` In this version of honesty, the out-of-bag observations for each tree are used as the honest (averaging) set. This setting also changes how predictions are constructed. When predicting for observations that are out-of-sample (using predict(..., aggregation = "average")), all the trees in the forest are used to construct predictions. When predicting for an observation that was in-sample (using predict(..., aggregation = "oob")), only the trees for which that observation was not in the averaging set are used to construct the prediction for that observation. aggregation="oob" (out-of-bag) ensures that the outcome value for an observation is never used to construct predictions for a given observation even when it is in sample. This property does not hold in standard honesty, which relies on an asymptotic subsampling argument. By default, when OOBhonest = TRUE, the out-of-bag observations for each tree are resamples with replacement to be used

for the honest (averaging) set. This results in a third set of observations that are left out of both the splitting and averaging set, we call these the double out-of-bag (doubleOOB) observations. In order to get the predictions of only the trees in which each observation fell into this doubleOOB set, one can run predict(... , aggregation = "doubleOOB"). In order to not do this second bootstrap sample, the doubleBootstrap flag can be set to FALSE.

`doubleBootstrap`
: The doubleBootstrap flag provides the option to resample with replacement from the out-of-bag observations set for each tree to construct the averaging set when using OOBhonest. If this is FALSE, the out-of-bag observations are used as the averaging set. By default this option is TRUE when running OOBhonest = TRUE. This option increases diversity across trees.

`nthread`
: Number of threads to train and predict the forest. The default number is 0 which represents using all cores.

`middleSplit`
: Indicator of whether the split value is takes the average of two feature values. If FALSE, it will take a point based on a uniform distribution between two feature values. (Default = FALSE)

`doubleTree`
: if the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. (Default = FALSE)

`linFeats`
: A vector containing the indices of which features to split linearly on when using linear penalized splits (defaults to use all numerical features).

`monotonicConstraints`
: Specifies monotonic relationships between the continuous features and the outcome. Supplied as a vector of length p with entries in 1,0,-1 which 1 indicating an increasing monotonic relationship, -1 indicating a decreasing monotonic relationship, and 0 indicating no constraint. Constraints supplied for categorical variable will be ignored.

`groups`
: A vector of factors specifying the group membership of each training observation. these groups are used in the aggregation when doing out of bag predictions in order to predict with only trees where the entire group was not used for aggregation. This allows the user to specify custom subgroups which will be used to create predictions which do not use any data from a common group to make predictions for any observation in the group. This can be used to create general custom resampling schemes, and provide predictions consistent with the Out-of-Group set.

`featureWeights`
: weights used when subsampling features for nodes above or at interactionDepth.

`deepFeatureWeights`
: weights used when subsampling features for nodes below interactionDepth.

`observationWeights`
: Denotes the weights for each training observation that determine how likely the observation is to be selected in each bootstrap sample. This option is not allowed when sampling is done without replacement.

`customSplitSample`
: List of vectors for user-defined splitting observations per tree. The vector at index i contains the indices of the sampled splitting observations, with replacement allowed, for tree i. This feature overrides other sampling parameters and must be set in conjunction with customAvgSample.

customAvgSample
: List of vectors for user-defined averaging observations per tree. The vector at index i contains the indices of the sampled splitting observations, with replacement allowed, for tree i. This feature overrides other sampling parameters and must be set in conjunction with customSplitSample.

customExcludeSample
: An optional list of vectors for user-defined excluded observations per tree. The vector at index i contains the indices of the excluded observations for tree i. An observation is considered excluded if it does not appear in the splitting or averaging set and has been explicitly withheld from being sampled for a tree. Excluded observations are not considered out-of-bag, so when we call predict with aggregation = "oob", when we predict for an observation, we will only use the predictions of trees in which the observation was in the customSplitSample (and neither in the customAvgSample nor the customExcludeSample). This parameter is optional even when customSplitSample and customAvgSample are set. It is also optional at the tree level, so can have fewer than ntree entries. When given fewer than ntree entries, for example K, the entries will be applied to the first K trees in the forest and the remaining trees will have no excludedSamples.

linear
: Indicator that enables Ridge penalized splits and linear aggregation functions in the leaf nodes. This is recommended for data with linear outcomes. For implementation details, see: https://arxiv.org/abs/1906.06463. Default is FALSE.

scale
: A parameter which indicates whether or not we want to scale and center the covariates and outcome before doing the regression. This can help with stability, so by default is TRUE.

hasNas
: indicates if there is any missingness in x.

naDirection
: Sets a default direction for missing values in each split node during training. It test placing all missing values to the left and right, then selects the direction that minimizes loss. If no missing values exist, then a default direction is randomly selected in proportion to the distribution of observations on the left and right. (Default = FALSE)

Value

A list of parameters after checking the selected parameters are valid.

`unscale_uncenter` *unscale_uncenter*

Description

Given a dataframe, un scale and un center the continous features

Usage

```
unscale_uncenter(x, categoricalFeatureCols, colMeans, colSd)
```

Arguments

x	A dataframe in order to be processed.
categoricalFeatureCols	A vector of the categorical features, we don't want to scale/center these. Should be 1-indexed.
colMeans	A vector of the means to add to each column.
colSd	A vector of the standard deviations to rescale each column with.

Value

A dataset x in it's original scaling

Index