Package: Rforestry (via r-universe)

October 19, 2026

Type Package

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

Version 0.11.1.0

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 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.

License GPL (>= 3) | file LICENSE

Encoding UTF-8

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

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 https://cran.r-universe.dev

Date/Publication 2025-03-15 23:40:02 UTC

RemoteUrl https://github.com/cran/Rforestry

RemoteRef HEAD

RemoteSha 499c58574057949c9a6183dfbff889d6415d0bb7

Contents

addTrees . . . 2
autoforestry . . . 3
autohonestRF . . . 4
compute_lp-forestry . . . 4
CppToR_translator . . . 5
forest_checker . . . 6
forestry . . . 6
forestry-class . . . 10
getOOB-forestry . . . 10
getOOBpreds-forestry . . . 11
getVI . . . 11
honestRF . . . 12
impute_features . . . 12
loadForestry . . . 13
make_savable . . . 13
multilayer-forestry . . . 14
plot-forestry . . . 17
predict-forestry . . . 18
predict-multilayer-forestry . . . 19
preprocess_testing . . . 20
preprocess_training . . . 20
relinkCPP_prt . . . 21
saveForestry . . . 21
testing_data_checker-forestry . . . 22
training_data_checker . . . 22

Index **25**

`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

`autoforestry` *autoforestry-forestry*

Description

autoforestry-forestry

Usage

```
autoforestry(
  x,
  y,
  sampsize = as.integer(nrow(x) * 0.75),
  num_iter = 1024,
  eta = 2,
  verbose = FALSE,
  seed = 24750371,
  nthread = 0
)
```

Arguments

`x`	A data frame of all training predictors.
`y`	A vector of all training responses.
`sampsize`	The size of total samples to draw for the training data.
`num_iter`	Maximum iterations/epochs per configuration. Default is 1024.
`eta`	Downsampling rate. Default value is 2.
`verbose`	if tuning process in verbose mode
`seed`	random seed
`nthread`	Number of threads to train and predict theforest. The default number is 0 which represents using all cores.

Value

A ‘forestry‘ object

autohonestRF *Honest Random Forest*

Description

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

Usage

```
autohonestRF(...)
```

Arguments

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

compute_lp-forestry *compute lp distances*

Description

return lp ditances of selected test observations.

Usage

```
compute_lp(object, feature.new, feature, p)
```

Arguments

object	A ‘forestry‘ object.
feature.new	A data frame of testing predictors.
feature	A string denoting the dimension for computing lp distances.
p	A positive real number determining the norm p-norm used.

Value

A vector 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)
predict(rf, x_test)

# Compute the l2 distances in the "Petal.Length" dimension
distances_2 <- compute_lp(object = rf,
                          feature.new = 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.

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

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 = 3,
  nodesizeAvg = 3,
  nodesizeStrictSpl = 1,
  nodesizeStrictAvg = 1,
  minSplitGain = 0,
  maxDepth = round(nrow(x)/2) + 1,
  interactionDepth = maxDepth,
  interactionVariables = numeric(0),
  featureWeights = NULL,
  deepFeatureWeights = NULL,
  observationWeights = NULL,
  splitratio = 1,
  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)),
  overfitPenalty = 1,
  doubleTree = 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 samplying without replacement, the default value is two-third 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 total number of features of the training data.
nodesizeSpl	Minimum observations contained in terminal nodes. The default value is 3.
nodesizeAvg	Minimum size of terminal nodes for averaging dataset. The default value is 3.
nodesizeStrictSpl	Minimum observations to follow strictly in terminal nodes. The default value is 1.
nodesizeStrictAvg	Minimum size of terminal nodes for averaging dataset to follow strictly. 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 probablities/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`
These denote the weights for each training observation which determines how likely the observation is to be selected in each bootstrap sample. This option is not allowed when sampling is done without replacement.

`splitratio` Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1, then essentially splitting dataset becomes the total entire sampled set and the averaging dataset is empty. If the ratio is 0, then the splitting data set is empty and all the data is used for the averaging data set (This is not a good usage however since there will be no data available for splitting).

`seed` random seed

`verbose` if training process 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 contains specifies the loss function according to which the splits of random forest should be made

`middleSplit` if the split value is taking 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` Fit the model with a ridge regression or not

`linFeats` Specify which features to split linearly on when using linear (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 relationship. Constraints supplied for categorical will be ignored.

`overfitPenalty` Value to determine how much to penalize magnitude of coefficients in ridge regression

`doubleTree` if the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. (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 dataset.

`savable` If TRUE, then RF is created in such a way that it can be saved and loaded using save(...) and load(...). Setting it to TRUE (default) will, however, take longer and it will use more memory. When training many RF, it makes a lot of 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

When training the forest, if a splitting feature is missing for an observation, we assign that observation to the child node which has an average y closer to the observed y of the observation with the missing feature, and record how many observations with missingness went to each child.

At predict time, if there were missing observations in a node at training time, we randomly assign an observation with a missing feature to a child node with probability proportional to the number of observations with a missing splitting variable that went to each child at training time. If there was no missingness at training time, we assign to the child nodes with probability proportional to the number of observations in each child node.

This procedure is a generalization of the usual recommended approach to missingness for forests—i.e., at each point add a decision to send the NAs to the left, right or to split on NA versus no NA. This usual recommendation is heuristically equivalent to adding an indicator for each feature plus a recoding of each missing variable where the missigness is the maximum and then the minimum observed value. This recommendation, however, allows the method to pickup time effects for when variables are missing because of the indicator. We, therefore, do not allow splitting on NAs. This should increase MSE in training but hopefully allows for better learning of universal relationships. Importantly, it is straightforward to show that our approach is weakly dominant in expected MSE to the always left or right approach. We should also note that almost no software package actually implements even the usual recommended approach—e.g., ranger does not.

In version 0.8.2.09, the procedure for identifying the best variable to split on when there is missing training data was modified. Previously candidate variables were evaluated by computing the MSE taken over all observations, including those for which the splitting variable was missing. In the current implementation we only use observations for which the splitting variable is not missing. The previous approach was biased towards splitting on variables with missingness because observations with a missing splitting variable are assigned to the leaf that minimized the MSE.

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)
weights = predict(rf, x_test, aggregation = "weightMatrix")$weightMatrix

weights %*% y_train
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,
          linear = TRUE
          )

predict(forest, x)
```

`forestry-class` *forestry class*

Description

'honestRF' class only exists for backwards compatibility reasons

---

`getOOB-forestry` *getOOB-forestry*

Description

Calculate the out-of-bag error of a given 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, noWarning)
```

Arguments

object	A trained model object of class "forestry".
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 increase in OOB for each shuffled feature for forest.

Usage

```
getVI(object, noWarning)
```

Arguments

object	A ‘forestry‘ object.
noWarning	flag to not display warnings

Note

No seed is passed to this function so it is not possible in the current implementation 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‘

impute_features *Feature imputation using random forests neigborhoods*

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 features for a point by computing the average, using neighborhoods weights, for all of the point's neighbors.

Usage

```
impute_features(
  object,
  feature.new,
  seed = round(runif(1) * 10000),
  use_mean_imputation_fallback = FALSE
)
```

Arguments

object	an object of class ‘forestry‘
feature.new	the feature data.frame we will impute
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 as NAs in the data frame returned by ‘impute_features‘.

Value

A data.frame that is feature.new 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)
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

---

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)
y_pred_before <- predict(forest, x)

forest <- make_savable(forest)
saveForestry(forest, file = "forest.Rda")
rm(forest)

forest <- loadForestry("forest.Rda")

y_pred_after <- predict(forest, x)
testthat::expect_equal(y_pred_before, y_pred_after, tolerance = 0.000001)
file.remove("forest.Rda")
```

---

multilayer-forestry *Multilayer forestry*

Description

Construct a gradient boosted random forest.

Usage

```
multilayerForestry(
  x,
  y,
  ntree = 500,
  nrounds = 1,
  eta = 0.3,
  replace = FALSE,
  sampsize = nrow(x),
  sample.fraction = NULL,
  mtry = ncol(x),
  nodesizeSpl = 3,
```

```
  nodesizeAvg = 3,
  nodesizeStrictSpl = max(round(nrow(x)/128), 1),
  nodesizeStrictAvg = max(round(nrow(x)/128), 1),
  minSplitGain = 0,
  maxDepth = 99,
  splitratio = 1,
  seed = as.integer(runif(1) * 1000),
  verbose = FALSE,
  nthread = 0,
  splitrule = "variance",
  middleSplit = TRUE,
  maxObs = length(y),
  linear = FALSE,
  linFeats = 0:(ncol(x) - 1),
  monotonicConstraints = rep(0, ncol(x)),
  featureWeights = rep(1, ncol(x)),
  deepFeatureWeights = featureWeights,
  observationWeights = NULL,
  overfitPenalty = 1,
  doubleTree = FALSE,
  reuseforestry = NULL,
  savable = TRUE,
  saveable = saveable
)
```

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.
nrounds	Number of iterations used for gradient boosting.
eta	Step size shrinkage used in gradient boosting update.
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 samplying without replacement, the default value is two-third 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 total number of features of the training data.
nodesizeSpl	Minimum observations contained in terminal nodes. The default value is 3.
nodesizeAvg	Minimum size of terminal nodes for averaging dataset. The default value is 3.
nodesizeStrictSpl	Minimum observations to follow strictly in terminal nodes. The default value is 1.

`nodesizeStrictAvg`
: Minimum size of terminal nodes for averaging dataset to follow strictly. 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.

`splitratio`
: Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1, then essentially splitting dataset becomes the total entire sampled set and the averaging dataset is empty. If the ratio is 0, then the splitting data set is empty and all the data is used for the averaging data set (This is not a good usage however since there will be no data available for splitting).

`seed`
: random seed

`verbose`
: if training process 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 contains specifies the loss function according to which the splits of random forest should be made

`middleSplit`
: if the split value is taking 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`
: Fit the model with a ridge regression or not

`linFeats`
: Specify which features to split linearly on when using linear (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 relationship. Constraints supplied for categorical will be ignored.

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

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

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

`overfitPenalty`
: Value to determine how much to penalize magnitude of coefficients in ridge regression

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

savable	If TRUE, then RF is created in such a way that it can be saved and loaded using save(...) and load(...). Setting it to TRUE (default) will, however, take longer and it will use more memory. When training many RF, it makes a lot of sense to set this to FALSE to save time and memory.
saveable	deprecated. Do not use.

Value

A 'multilayerForestry' object.

`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 visulaized.
print.meta_dta	Should the data for the plot be printed?
beta.char.len	The length of the beta values in leaf node representation.
...	additional arguments that are not used.

Examples

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

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

ridge_rf <- forestry(
  x = iris[,-1],
  y = iris[, 1],
  replace = FALSE,
  nodesizeStrictSpl = 10,
  mtry = 4,
  ntree = 10,
```

```
  minSplitGain = .004,
  linear = TRUE,
  overfitPenalty = 1.65,
  linFeats = 1:2)

plot(x = ridge_rf)
plot(x = ridge_rf, tree.id = 2)
plot(x = ridge_rf, tree.id = 10)
```

---

predict-forestry *predict-forestry*

Description

Return the prediction from the forest.

Usage

```
## S3 method for class 'forestry'
predict(
  object,
  feature.new,
  aggregation = "average",
  seed = as.integer(runif(1) * 10000),
  ...
)
```

Arguments

object	A ‘forestry‘ object.
feature.new	A data frame of testing predictors.
aggregation	How the individual tree predictions are aggregated: ‘average‘ returns the mean of all trees in the forest; ‘weightMatrix‘ returns a list consisting of "weightMatrix", the adaptive nearest neighbor weights used to construct the predictions; "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 feature.new 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.
seed	random seed
...	additional arguments.

Value

A vector of predicted responses.

`predict-multilayer-forestry`
predict-multilayer-forestry

Description

Return the prediction from the forest.

Usage

```
## S3 method for class 'multilayerForestry'
predict(
  object,
  feature.new,
  aggregation = "average",
  seed = as.integer(runif(1) * 10000),
  ...
)
```

Arguments

`object`	A ‘multilayerForestry‘ object.
`feature.new`	A data frame of testing predictors.
`aggregation`	How shall the leaf be aggregated. The default is to return the mean of the leave ‘average‘. Other options are ‘weightMatrix‘.
`seed`	random seed
`...`	additional arguments.

Value

A vector of predicted responses.

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 encoding 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' or class 'multilayerForestry'

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

testing_data_checker-forestry *Test data check*

Description

Check the testing data to do prediction

Usage

```
testing_data_checker(object, feature.new, hasNas)
```

Arguments

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

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,
  nthread,
  middleSplit,
  doubleTree,
```

```
  linFeats,
  monotonicConstraints,
  featureWeights,
  deepFeatureWeights,
  observationWeights,
  linear,
  hasNas
)
```

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 samplying without replacement, the default value is two-third 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 total number of features of the training data.
nodesizeSpl	Minimum observations contained in terminal nodes. The default value is 3.
nodesizeAvg	Minimum size of terminal nodes for averaging dataset. The default value is 3.
nodesizeStrictSpl	Minimum observations to follow strictly in terminal nodes. The default value is 1.
nodesizeStrictAvg	Minimum size of terminal nodes for averaging dataset to follow strictly. 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, then essentially splitting dataset becomes the total entire sampled set and the averaging dataset is empty. If the ratio is 0, then the splitting data set is empty and all the data is used for the averaging data set (This is not a good usage however since there will be no data available for splitting).
nthread	Number of threads to train and predict the forest. The default number is 0 which represents using all cores.
middleSplit	if the split value is taking 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` Specify which features to split linearly on when using linear (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 relationship. Constraints supplied for categorical will be ignored.

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

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

`observationWeights`
These denote the weights for each training observation which determines how likely the observation is to be selected in each bootstrap sample. This option is not allowed when sampling is done without replacement.

`linear` Fit the model with a ridge regression or not

`hasNas` indicates if there is any missingness in x.

Index

addTrees, 2
autoforestry, 3
autohonestRF, 4

compute_lp *(*compute_lp-forestry*)*, 4
compute_lp-forestry, 4
CppToR_translator, 5

forest_checker, 6
forestry, 6, *11*
forestry-class, 10

getOOB *(*getOOB-forestry*)*, 10
getOOB,forestry-method *(*getOOB-forestry*)*, 10
getOOB-forestry, 10
getOOBpreds *(*getOOBpreds-forestry*)*, 11
getOOBpreds-forestry, 11
getVI, 11

honestRF, 12

impute_features, 12

loadForestry, 13

make_savable, 13
make_savable,forestry-method *(*make_savable*)*, 13
multilayer-forestry, 14
multilayerForestry *(*multilayer-forestry*)*, 14

plot-forestry, 17
plot.forestry *(*plot-forestry*)*, 17
predict-forestry, 18
predict-multilayer-forestry, 19
predict.forestry *(*predict-forestry*)*, 18
predict.multilayerForestry *(*predict-multilayer-forestry*)*, 19

preprocess_testing, 20
preprocess_training, 20

relinkCPP_prt, 21

saveForestry, 21

testing_data_checker *(*testing_data_checker-forestry*)*, 22
testing_data_checker-forestry, 22
training_data_checker, 22