

# Package ‘Rforestry’

June 3, 2021

**Type** Package

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

**Version** 0.9.0.40

**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](https://arxiv.org/abs/1906.06463)>.

**License** GPL (>= 3)

**Encoding** UTF-8

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

**LinkingTo** Rcpp, RcppArmadillo, RcppThread

**RoxygenNote** 7.1.1

**SystemRequirements** C++11

**Suggests** testthat, knitr, rmarkdown, mvtnorm

**Collate** 'R\_preprocessing.R' 'RcppExports.R' 'forestry.R'  
'adaptive\_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],  
Edward Liu [aut],  
Allen Tang [aut],  
Jasjeet Sekhon [aut]

**Repository** CRAN

**Date/Publication** 2021-06-03 06:10:04 UTC

## R topics documented:

adaptiveForestry-forestry . . . . .	2
addTrees . . . . .	6
autoforestry . . . . .	7
autohonestRF . . . . .	8
compute_lp-forestry . . . . .	8
CppToR_translator . . . . .	9
forestry . . . . .	10
forestry-class . . . . .	14
forest_checker . . . . .	14
getCI . . . . .	15
getOOB-forestry . . . . .	16
getOOBpreds-forestry . . . . .	16
getSplitProps-forestry . . . . .	17
getVI . . . . .	18
honestRF . . . . .	18
impute_features . . . . .	19
loadForestry . . . . .	20
make_savable . . . . .	20
multilayer-forestry . . . . .	21
plot-forestry . . . . .	24
predict-adaptiveForestry . . . . .	25
predict-forestry . . . . .	27
predict-multilayer-forestry . . . . .	28
preprocess_testing . . . . .	29
preprocess_training . . . . .	30
relinkCPP_prt . . . . .	30
saveForestry . . . . .	31
testing_data_checker-forestry . . . . .	31
training_data_checker . . . . .	32
<b>Index</b>	<b>35</b>

---

adaptiveForestry-forestry

*forestry with adaptive featureWeights*

---

### Description

This is an experimental function where we run forestry in two stages, first estimating the feature weights by calculating the relative splitting proportions of each feature using a small forest, and then growing a much bigger forest using the first forest splitting proportions as the featureWeights in the second forest.

**Usage**

```

adaptiveForestry(
  x,
  y,
  ntree = 500,
  ntree.first = 25,
  ntree.second = 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,
  splitratio = 1,
  OOBhonest = 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)),
  monotoneAvg = FALSE,
  overfitPenalty = 1,
  doubleTree = FALSE,
  reuseforestry = NULL,
  savable = TRUE,
  saveable = TRUE
)

```

**Arguments**

<code>x</code>	A data frame of all training predictors.
<code>y</code>	A vector of all training responses.
<code>ntree</code>	The number of trees to grow in the forest. The default value is 500.
<code>ntree.first</code>	The number of trees to grow in the first forest when trying to determine which features are important.

<code>nTree.second</code>	The number of features to use in the second stage when we grow a second forest using the weights of the first stage.
<code>replace</code>	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
<code>sampsize</code>	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.
<code>sample.fraction</code>	If this is given, then <code>sampsize</code> is ignored and set to be <code>round(length(y) * sample.fraction)</code> . It must be a real number between 0 and 1
<code>mtry</code>	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.
<code>nodesizeSpl</code>	Minimum observations contained in terminal nodes. The default value is 5.
<code>nodesizeAvg</code>	Minimum size of terminal nodes for averaging dataset. The default value is 5.
<code>nodesizeStrictSpl</code>	Minimum observations to follow strictly in terminal nodes. The default value is 1.
<code>nodesizeStrictAvg</code>	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 <code>nodesizeStrictAvg</code> averaging observations in either child node will be rejected, ensuring every leaf node also has at least <code>nodesizeStrictAvg</code> averaging observations. The default value is 1.
<code>minSplitGain</code>	Minimum loss reduction to split a node further in a tree.
<code>maxDepth</code>	Maximum depth of a tree. The default value is 99.
<code>interactionDepth</code>	All splits at or above interaction depth must be on variables that are not weighting variables (as provided by the <code>interactionVariables</code> argument).
<code>interactionVariables</code>	Indices of weighting variables.
<code>featureWeights</code>	(optional) vector of sampling probabilities/weights for each feature used when subsampling <code>mtry</code> features at each node above or at <code>interactionDepth</code> . The default is to use uniform probabilities.
<code>deepFeatureWeights</code>	Used in place of <code>featureWeights</code> for splits below <code>interactionDepth</code> .
<code>observationWeights</code>	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.
<code>splitratio</code>	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.
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: <a href="https://arxiv.org/abs/1906.06463">https://arxiv.org/abs/1906.06463</a> . 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.
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.
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 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.

### Details

adaptiveForestry

### Value

Two forestry objects, the first forest, and the adaptive forest, as well as the splitting proportions used to grow the second forest.

### 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 <- adaptiveForestry(x = x_train,
                      y = y_train,
                      ntree.first = 25,
                      ntree.second = 500,
                      nthread = 2)
predict(rf@second.forest, x_test)
```

---

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 the forest. The default number is 0 which represents using all cores.

**Value**

A 'forestry' object

---

autohonestRF	<i>Honest Random Forest</i>
--------------	-----------------------------

---

**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’

**Value**

A ‘forestry’ object

---

compute_lp-forestry	<i>compute lp distances</i>
---------------------	-----------------------------

---

**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)
```

**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.

**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 12 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,  
  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)),  
  monotoneAvg = FALSE,  
  overfitPenalty = 1,  
  doubleTree = FALSE,  
  reuseforestry = NULL,  
  savable = TRUE,  
  saveable = TRUE  
)
```

**Arguments**

<code>x</code>	A data frame of all training predictors.
<code>y</code>	A vector of all training responses.
<code>ntree</code>	The number of trees to grow in the forest. The default value is 500.
<code>replace</code>	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
<code>sampsize</code>	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.
<code>sample.fraction</code>	If this is given, then <code>sampsize</code> is ignored and set to be <code>round(length(y) * sample.fraction)</code> . It must be a real number between 0 and 1
<code>mtry</code>	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.
<code>nodesizeSpl</code>	Minimum observations contained in terminal nodes. The default value is 5.
<code>nodesizeAvg</code>	Minimum size of terminal nodes for averaging dataset. The default value is 5.
<code>nodesizeStrictSpl</code>	Minimum observations to follow strictly in terminal nodes. The default value is 1.
<code>nodesizeStrictAvg</code>	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 <code>nodesizeStrictAvg</code> averaging observations in either child node will be rejected, ensuring every leaf node also has at least <code>nodesizeStrictAvg</code> averaging observations. The default value is 1.
<code>minSplitGain</code>	Minimum loss reduction to split a node further in a tree.
<code>maxDepth</code>	Maximum depth of a tree. The default value is 99.
<code>interactionDepth</code>	All splits at or above interaction depth must be on variables that are not weighting variables (as provided by the <code>interactionVariables</code> argument).
<code>interactionVariables</code>	Indices of weighting variables.
<code>featureWeights</code>	(optional) vector of sampling probabilities/weights for each feature used when subsampling <code>mtry</code> features at each node above or at <code>interactionDepth</code> . The default is to use uniform probabilities.
<code>deepFeatureWeights</code>	Used in place of <code>featureWeights</code> for splits below <code>interactionDepth</code> .
<code>observationWeights</code>	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.

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.
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: <a href="https://arxiv.org/abs/1906.06463">https://arxiv.org/abs/1906.06463</a> . 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.

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.
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 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	<i>forestry class</i>
----------------	-----------------------

---

### Description

'honestRF' class only exists for backwards compatibility reasons

---

forest_checker	<i>Checks if forestry object has valid pointer for C++ object.</i>
----------------	--

---

### 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.

---

getCI	<i>getCI-forestry</i>
-------	-----------------------

---

### Description

For a new set of features, calculate the confidence intervals for each new observation.

### Usage

```
getCI(
  object,
  newdata,
  level = 0.95,
  B = 100,
  method = "OOB-conformal",
  noWarning = FALSE
)
```

### Arguments

object	A 'forestry' object.
newdata	A set of new observations for which we want to predict the outcomes and use confidence intervals.
level	The confidence level at which we want to make our intervals. Default is to use .95 which corresponds to 95 percentile confidence intervals.
B	Number of bootstrap draws to use when using method = "OOB-bootstrap"
method	A flag for the different ways to create the confidence intervals. Right now we have two ways of doing this. One is the 'OOB-bootstrap' flag which uses many bootstrap pulls from the set of OOB trees then with these different pulls, we use the set of trees to predict for the new feature and give the confidence set over the many bootstrap draws. The other method- 'OOB-conformal'- creates intervals by taking the set of doubleOOB trees for each observation, and using the predictions of these trees to give conformal intervals. So for an observation $obs_i$ , let $S_i$ be the set of trees for which $obs_i$ was in neither the splitting set nor the averaging set (or the set of trees for which $obs_i$ was "doubleOOB"), we then predict for $obs_i$ with only the trees in $S_i$ . <code>doubleOOB_tree_preds &lt;- predict(S_i, obs_i)</code> : Then <code>CI(obs_i, alpha = .95) = quantile(doubleOOB_tree_preds - y_i, probs = .95)</code> . The 'local-conformal' option takes the residuals of each training point (using) OOB predictions, and then uses the weights of the random forest to determine the quantiles of the residuals in the local neighborhood of the predicted point. Default is 'OOB-conformal'.
noWarning	flag to not display warnings

### Value

The confidence intervals for each observation in newdata.

---

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](#)

---

`getSplitProps-forestry`

*getSplitProps-forestry*

---

**Description**

Retrieves the proportion of splits for each feature in the given forestry object. These proportions are calculated as the number of splits on feature *i* in the entire forest over total the number of splits in the forest.

**Usage**

```
getSplitProps(object)
```

**Arguments**

`object`            A trained model object of class "forestry".

**Value**

A vector of length equal to the number of columns

**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)
```

**Arguments**

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

**Value**

The variable importance of the forest.

**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'
-----	--

**Value**

A 'forestry' object

---

impute_features	<i>Feature imputation using random forests neighborhoods</i>
-----------------	--

---

### 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\_ptr' 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"))
```

---

multilayer-forestry    *Multilayer forestry*

---

**Description**

Construct a gradient boosted ensemble with random forest base learners.

**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,
  OOBhonest = FALSE,
  doubleBootstrap = if (OOBhonest) TRUE else FALSE,
  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)),
monotoneAvg = FALSE,
featureWeights = rep(1, ncol(x)),
deepFeatureWeights = featureWeights,
observationWeights = NULL,
overfitPenalty = 1,
doubleTree = FALSE,
reuseforestry = NULL,
savable = TRUE,
saveable = saveable
)

```

### Arguments

<code>x</code>	A data frame of all training predictors.
<code>y</code>	A vector of all training responses.
<code>ntree</code>	The number of trees to grow in the forest. The default value is 500.
<code>nrounds</code>	Number of iterations used for gradient boosting.
<code>eta</code>	Step size shrinkage used in gradient boosting update.
<code>replace</code>	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
<code>sampsize</code>	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.
<code>sample.fraction</code>	If this is given, then <code>sampsize</code> is ignored and set to be <code>round(length(y) * sample.fraction)</code> . It must be a real number between 0 and 1
<code>mtry</code>	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.
<code>nodesizeSpl</code>	Minimum observations contained in terminal nodes. The default value is 5.
<code>nodesizeAvg</code>	Minimum size of terminal nodes for averaging dataset. The default value is 5.
<code>nodesizeStrictSpl</code>	Minimum observations to follow strictly in terminal nodes. The default value is 1.
<code>nodesizeStrictAvg</code>	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.
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.
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: <a href="https://arxiv.org/abs/1906.06463">https://arxiv.org/abs/1906.06463</a> . 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.
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.
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.
overfitPenalty	Value to determine how much to penalize the magnitude of coefficients in ridge regression when using linear splits.
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 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 'multilayerForestry' object.

---

plot-forestry	<i>visualize a tree</i>
---------------	-------------------------

---

**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**

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

**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)

ridge_rf <- forestry(
  x = iris[,-1],
  y = iris[, 1],
  replace = FALSE,
  nodesizeStrictSpl = 10,
  mtry = 4,
  ntree = 1000,
  minSplitGain = .004,
  linear = TRUE,
  nthread = 2,
  overfitPenalty = 1.65,
  linFeats = 1:2)

plot(x = ridge_rf)
plot(x = ridge_rf, tree.id = 2)
plot(x = ridge_rf, tree.id = 1000)
```

**Description**

Return the prediction from the forest.

**Usage**

```
## S3 method for class 'adaptiveForestry'
predict(
  object,
  newdata,
  aggregation = "average",
  seed = as.integer(runif(1) * 10000),
  nthread = 0,
  exact = NULL,
  weighting = NULL,
  ...
)
```

**Arguments**

object	An 'adaptiveForestry' 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; '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
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.
weighting	This should be a number between 0 and 1 indicating the weight with which to use the predictions of the two forests. This specifically specifies the weight given to the second.forest object. The predictions are given by $\text{weighting} * \text{predict}(\text{object}@\text{second.forest}) + (1 - \text{weighting}) * \text{predict}(\text{object}@\text{first.forest})$ . Defaults to NULL, and in this case, $\text{weighting} = \text{n.tree.second} / (\text{n.tree.first} + \text{n.tree.second})$ .
...	additional arguments.

**Value**

A vector of predicted responses.

---

predict-forestry	<i>predict-forestry</i>
------------------	-------------------------

---

**Description**

Return the prediction from the forest.

**Usage**

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

**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; 'weightMatrix' returns a list consisting of "weightMatrix", the adaptive nearest neighbor weights used to construct the predictions; 'terminalNodes' also returns the weightMatrix, as well as "terminalNodes", a matrix where the <i>i</i> th entry of the <i>j</i> th column is the index of the leaf node to which the <i>i</i> th observation is assigned in the <i>j</i> th tree; and "sparse", a matrix where the <i>i</i> th entry in the <i>j</i> th column is 1 if the <i>i</i> th observation in newdata is assigned to the <i>j</i> th 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).

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:nree 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 different 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: $\text{predict}(\dots, \text{trees} = \text{c}(1,2,2)) = (\text{predict}(\dots, \text{trees} = \text{c}(1)) + 2 * \text{predict}(\dots, \text{trees} = \text{c}(2))) / 3$ . note we must have exact = TRUE, and aggregation = "average" to use tree indices.
...	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,
  newdata,
  aggregation = "average",
  seed = as.integer(runif(1) * 10000),
  nthread = 0,
  exact = NULL,
  ...
)
```

**Arguments**

object	A 'multilayerForestry' object.
newdata	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' which returns the adaptive nearest neighbor weights used to construct the predictions.
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.
...	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 dataset 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 'forestry' 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'

**Value**

Relinks the pointer to the correct C++ object.

---

saveForestry	<i>save RF</i>
--------------	----------------

---

**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.

---

testing_data_checker-forestry	<i>Test data check</i>
-------------------------------	------------------------

---

**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 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,  
  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 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.
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)

<code>doubleTree</code>	if the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. (Default = FALSE)
<code>linFeats</code>	A vector containing the indices of which features to split linearly on when using linear penalized splits (defaults to use all numerical features).
<code>monotonicConstraints</code>	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.
<code>featureWeights</code>	weights used when subsampling features for nodes above or at <code>interactionDepth</code> .
<code>deepFeatureWeights</code>	weights used when subsampling features for nodes below <code>interactionDepth</code> .
<code>observationWeights</code>	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.
<code>linear</code>	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: <a href="https://arxiv.org/abs/1906.06463">https://arxiv.org/abs/1906.06463</a> . Default is FALSE.
<code>hasNas</code>	indicates if there is any missingness in $x$ .

**Value**

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

# Index

adaptiveForestry  
    (adaptiveForestry-forestry), 2  
adaptiveForestry-forestry, 2  
addTrees, 6  
autoforestry, 7  
autohonestRF, 8  
  
compute\_lp (compute\_lp-forestry), 8  
compute\_lp-forestry, 8  
CppToR\_translator, 9  
  
forest\_checker, 14  
forestry, 10, 17  
forestry-class, 14  
  
getCI, 15  
getOOB (getOOB-forestry), 16  
getOOB, forestry-method  
    (getOOB-forestry), 16  
getOOB-forestry, 16  
getOOBpreds (getOOBpreds-forestry), 16  
getOOBpreds-forestry, 16  
getSplitProps (getSplitProps-forestry),  
    17  
getSplitProps-forestry, 17  
getVI, 18  
  
honestRF, 18  
  
impute\_features, 19  
  
loadForestry, 20  
  
make\_savable, 20  
make\_savable, forestry-method  
    (make\_savable), 20  
multilayer-forestry, 21  
multilayerForestry  
    (multilayer-forestry), 21  
  
plot-forestry, 24  
  
plot.forestry (plot-forestry), 24  
predict-adaptiveForestry, 25  
predict-forestry, 27  
predict-multilayer-forestry, 28  
predict.adaptiveForestry  
    (predict-adaptiveForestry), 25  
predict.forestry (predict-forestry), 27  
predict.multilayerForestry  
    (predict-multilayer-forestry),  
    28  
preprocess\_testing, 29  
preprocess\_training, 30  
  
relinkCPP\_prt, 30  
  
saveForestry, 31  
  
testing\_data\_checker  
    (testing\_data\_checker-forestry),  
    31  
testing\_data\_checker-forestry, 31  
training\_data\_checker, 32