Package 'ranger'

November 8, 2024

Type Package

Title A Fast Implementation of Random Forests

Version 0.17.0

Date 2024-11-08

Description A fast implementation of Random Forests, particularly suited for high dimensional data. Ensembles of classification, regression, survival and probability prediction trees are supported. Data from genome-wide association studies can be analyzed efficiently. In addition to data frames, datasets of class 'gwaa.data' (R package 'GenABEL') and 'dgCMatrix' (R package 'Matrix') can be directly analyzed.

License GPL-3

Imports Rcpp (>= 0.11.2), Matrix

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.1)

Suggests survival, testthat

Encoding UTF-8

RoxygenNote 7.3.2

URL https://imbs-hl.github.io/ranger/,
https://github.com/imbs-hl/ranger

BugReports https://github.com/imbs-hl/ranger/issues

NeedsCompilation yes

Author Marvin N. Wright [aut, cre],
Stefan Wager [ctb],
Philipp Probst [ctb]

Maintainer Marvin N. Wright <cran@wrig.de>

Repository CRAN

Date/Publication 2024-11-08 07:40:02 UTC

Contents

`csrf` *Case-specific random forests.*

Description

In case-specific random forests (CSRF), random forests are built specific to the cases of interest. Instead of using equal probabilities, the cases are weighted according to their difference to the case of interest.

Usage

```
csrf(
  formula,
  training_data,
  test_data,
  params1 = list(),
  params2 = list(),
  verbose = FALSE
)
```

Arguments

formula	Object of class formula or character describing the model to fit.
training_data	Training data of class data.frame.
test_data	Test data of class data.frame.
params1	Parameters for the proximity random forest grown in the first step.
params2	Parameters for the prediction random forests grown in the second step.
verbose	Logical indicating whether or not to print computation progress.

Details

The algorithm consists of 3 steps:

1. Grow a random forest on the training data
2. For each observation of interest (test data), the weights of all training observations are computed by counting the number of trees in which both observations are in the same terminal node.
3. For each test observation, grow a weighted random forest on the training data, using the weights obtained in step 2. Predict the outcome of the test observation as usual.

In total, n+1 random forests are grown, where n is the number observations in the test dataset. For details, see Xu et al. (2014).

Value

Predictions for the test dataset.

Author(s)

Marvin N. Wright

References

Xu, R., Nettleton, D. & Nordman, D.J. (2014). Case-specific random forests. J Comp Graph Stat 25:49-65. doi:10.1080/10618600.2014.983641.

Examples

```
## Split in training and test data
train.idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]

## Run case-specific RF
csrf(Species ~ ., training_data = iris.train, test_data = iris.test,
     params1 = list(num.trees = 50, mtry = 4),
     params2 = list(num.trees = 5))
```

deforest *Deforesting a random forest*

Description

The main purpose of this function is to allow for post-processing of ensembles via L2 regularized regression (i.e., the LASSO), as described in Friedman and Popescu (2003). The basic idea is to use the LASSO to post-process the predictions from the individual base learners in an ensemble (i.e., decision trees) in the hopes of producing a much smaller model without sacrificing much in the way of accuracy, and in some cases, improving it. Friedman and Popescu (2003) describe conditions under which tree-based ensembles, like random forest, can potentially benefit from such post-processing (e.g., using shallower trees trained on much smaller samples of the training data without replacement). However, the computational benefits of such post-processing can only be realized if the base learners "zeroed out" by the LASSO can actually be removed from the original ensemble, hence the purpose of this function. A complete example using `ranger` can be found at https://github.com/imbs-hl/ranger/issues/568.

Usage

```
deforest(object, which.trees = NULL, ...)

## S3 method for class 'ranger'
deforest(object, which.trees = NULL, warn = TRUE, ...)
```

Arguments

`object`	A fitted random forest (e.g., a `ranger` object).
`which.trees`	Vector giving the indices of the trees to remove.
`...`	Additional (optional) arguments. (Currently ignored.)
`warn`	Logical indicating whether or not to warn users that some of the standard output of a typical `ranger` object or no longer available after deforestation. Default is `TRUE`.

Value

An object of class `"deforest.ranger"`; essentially, a `ranger` object with certain components replaced with `NA`s (e.g., out-of-bag (OOB) predictions, variable importance scores (if requested), and OOB-based error metrics).

Note

This function is a generic and can be extended by other packages.

Author(s)

Brandon M. Greenwell

References

Friedman, J. and Popescu, B. (2003). Importance sampled learning ensembles, Technical report, Stanford University, Department of Statistics. https://jerryfriedman.su.domains/ftp/isle.pdf.

Examples

```
## Example of deforesting a random forest
rfo <- ranger(Species ~ ., data = iris, probability = TRUE, num.trees = 100)
dfo <- deforest(rfo, which.trees = c(1, 3, 5))
dfo  # same as `rfo` but with trees 1, 3, and 5 removed

## Sanity check
preds.rfo <- predict(rfo, data = iris, predict.all = TRUE)$predictions
preds.dfo <- predict(dfo, data = iris, predict.all = TRUE)$predictions
identical(preds.rfo[, , -c(1, 3, 5)], y = preds.dfo)
```

---

getTerminalNodeIDs *Get terminal node IDs (deprecated)*

Description

This function is deprecated. Please use predict() with `type = "terminalNodes"` instead. This function calls predict() now.

Usage

```
getTerminalNodeIDs(rf, dat)
```

Arguments

`rf`	`ranger` object.
`dat`	New dataset. Terminal node IDs for this dataset are obtained.

Value

Matrix with terminal nodeIDs for all observations in dataset and trees.

Examples

```
rf <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
getTerminalNodeIDs(rf, iris)
```

`holdoutRF` *Hold-out random forests*

Description

Grow two random forests on two cross-validation folds. Instead of out-of-bag data, the other fold is used to compute permutation importance. Related to the novel permutation variable importance by Janitza et al. (2015).

Usage

```
holdoutRF(...)
```

Arguments

`...`	All arguments are passed to `ranger()` (except `importance`, `case.weights`, `replace` and `holdout`.).

Value

Hold-out random forests with variable importance.

Author(s)

Marvin N. Wright

References

Janitza, S., Celik, E. & Boulesteix, A.-L., (2015). A computationally fast variable importance test for random forests for high-dimensional data. Adv Data Anal Classif doi:10.1007/s116340160276-4.

See Also

`ranger`

`hshrink` *Hierarchical shrinkage*

Description

Apply hierarchical shrinkage to a ranger object. Hierarchical shrinkage is a regularization technique that recursively shrinks node predictions towards parent node predictions. For details see Agarwal et al. (2022).

Usage

```
hshrink(rf, lambda)
```

Arguments

`rf`	ranger object, created with `node.stats = TRUE`.
`lambda`	Non-negative shrinkage parameter.

Value

The ranger object is modified in-place.

Author(s)

Marvin N. Wright

References

- Agarwal, A., Tan, Y.S., Ronen, O., Singh, C. & Yu, B. (2022). Hierarchical Shrinkage: Improving the accuracy and interpretability of tree-based models. Proceedings of the 39th International Conference on Machine Learning, PMLR 162:111-135.

Examples

```
## Hierarchical shrinkage for a probablity forest
rf <- ranger(Species ~ ., iris, node.stats = TRUE, probability = TRUE)
hshrink(rf, lambda = 5)
```

importance.ranger *ranger variable importance*

Description

Extract variable importance of ranger object.

Usage

```
## S3 method for class 'ranger'
importance(x, ...)
```

Arguments

x	ranger object.
...	Further arguments passed to or from other methods.

Value

Variable importance measures.

Author(s)

Marvin N. Wright

See Also

ranger

importance_pvalues *ranger variable importance p-values*

Description

Compute variable importance with p-values. For high dimensional data, the fast method of Janitza et al. (2016) can be used. The permutation approach of Altmann et al. (2010) is computationally intensive but can be used with all kinds of data. See below for details.

Usage

```
importance_pvalues(
  x,
  method = c("janitza", "altmann"),
  num.permutations = 100,
  formula = NULL,
  data = NULL,
  ...
)
```

Arguments

x	ranger or holdoutRF object.
method	Method to compute p-values. Use "janitza" for the method by Janitza et al. (2016) or "altmann" for the non-parametric method by Altmann et al. (2010).
num.permutations	Number of permutations. Used in the "altmann" method only.
formula	Object of class formula or character describing the model to fit. Used in the "altmann" method only.
data	Training data of class data.frame or matrix. Used in the "altmann" method only.
...	Further arguments passed to ranger(). Used in the "altmann" method only.

Details

The method of Janitza et al. (2016) uses a clever trick: With an unbiased variable importance measure, the importance values of non-associated variables vary randomly around zero. Thus, all non-positive importance values are assumed to correspond to these non-associated variables and they are used to construct a distribution of the importance under the null hypothesis of no association to the response. Since only the non-positive values of this distribution can be observed, the positive values are created by mirroring the negative distribution. See Janitza et al. (2016) for details.

The method of Altmann et al. (2010) uses a simple permutation test: The distribution of the importance under the null hypothesis of no association to the response is created by several replications of permuting the response, growing an RF and computing the variable importance. The authors recommend 50-100 permutations. However, much larger numbers have to be used to estimate more precise p-values. We add 1 to the numerator and denominator to avoid zero p-values.

Value

Variable importance and p-value for each variable.

Author(s)

Marvin N. Wright

References

Janitza, S., Celik, E. & Boulesteix, A.-L., (2016). A computationally fast variable importance test for random forests for high-dimensional data. Adv Data Anal Classif doi:10.1007/s116340160276-4.

Altmann, A., Tolosi, L., Sander, O. & Lengauer, T. (2010). Permutation importance: a corrected feature importance measure, Bioinformatics 26:1340-1347.

See Also

ranger

Examples

```
## Janitza's p-values with corrected Gini importance
n <- 50
p <- 400
dat <- data.frame(y = factor(rbinom(n, 1, .5)), replicate(p, runif(n)))
rf.sim <- ranger(y ~ ., dat, importance = "impurity_corrected")
importance_pvalues(rf.sim, method = "janitza")

## Permutation p-values
## Not run:
rf.iris <- ranger(Species ~ ., data = iris, importance = 'permutation')
importance_pvalues(rf.iris, method = "altmann", formula = Species ~ ., data = iris)

## End(Not run)
```

`parse.formula` *Parse formula*

Description

Parse formula and return dataset containing selected columns. Interactions are supported for numerical columns only. An interaction column is the product of all interacting columns.

Usage

```
parse.formula(formula, data, env = parent.frame())
```

Arguments

`formula`	Object of class `formula` or `character` describing the model to fit.
`data`	Training data of class `data.frame`.
`env`	The environment in which the left hand side of `formula` is evaluated.

Value

Dataset including selected columns and interactions.

predict.ranger *Ranger prediction*

Description

Prediction with new data and a saved forest from Ranger.

Usage

```
## S3 method for class 'ranger'
predict(
  object,
  data = NULL,
  predict.all = FALSE,
  num.trees = object$num.trees,
  type = "response",
  se.method = "infjack",
  quantiles = c(0.1, 0.5, 0.9),
  what = NULL,
  seed = NULL,
  num.threads = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

object	Ranger ranger object.
data	New test data of class data.frame or gwaa.data (GenABEL).
predict.all	Return individual predictions for each tree instead of aggregated predictions for all trees. Return a matrix (sample x tree) for classification and regression, a 3d array for probability estimation (sample x class x tree) and survival (sample x time x tree).
num.trees	Number of trees used for prediction. The first num.trees in the forest are used.
type	Type of prediction. One of 'response', 'se', 'terminalNodes', 'quantiles' with default 'response'. See below for details.
se.method	Method to compute standard errors. One of 'jack', 'infjack' with default 'infjack'. Only applicable if type = 'se'. See below for details.
quantiles	Vector of quantiles for quantile prediction. Set type = 'quantiles' to use.
what	User specified function for quantile prediction used instead of quantile. Must return numeric vector, see examples.
seed	Random seed. Default is NULL, which generates the seed from R. Set to 0 to ignore the R seed. The seed is used in case of ties in classification mode.
num.threads	Number of threads. Use 0 for all available cores. Default is 2 if not set by options/environment variables (see below).

verbose	Verbose output on or off.
...	further arguments passed to or from other methods.

Details

For type = 'response' (the default), the predicted classes (classification), predicted numeric values (regression), predicted probabilities (probability estimation) or survival probabilities (survival) are returned. For type = 'se', the standard error of the predictions are returned (regression only). The jackknife-after-bootstrap or infinitesimal jackknife for bagging is used to estimate the standard errors based on out-of-bag predictions. See Wager et al. (2014) for details. For type = 'terminalNodes', the IDs of the terminal node in each tree for each observation in the given dataset are returned. For type = 'quantiles', the selected quantiles for each observation are estimated. See Meinshausen (2006) for details.

If type = 'se' is selected, the method to estimate the variances can be chosen with se.method. Set se.method = 'jack' for jackknife-after-bootstrap and se.method = 'infjack' for the infinitesimal jackknife for bagging.

For classification and predict.all = TRUE, a factor levels are returned as numerics. To retrieve the corresponding factor levels, use rf$forest$levels, if rf is the ranger object.

By default, ranger uses 2 threads. The default can be changed with: (1) num.threads in ranger/predict call, (2) environment variable R_RANGER_NUM_THREADS, (3) options(ranger.num.threads = N), (4) options(Ncpus = N), with precedence in that order.

Value

Object of class ranger.prediction with elements

predictions	Predicted classes/values (only for classification and regression)
unique.death.times	Unique death times (only for survival).
chf	Estimated cumulative hazard function for each sample (only for survival).
survival	Estimated survival function for each sample (only for survival).
num.trees	Number of trees.
num.independent.variables	Number of independent variables.
treetype	Type of forest/tree. Classification, regression or survival.
num.samples	Number of samples.

Author(s)

Marvin N. Wright

References

- Wright, M. N. & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.
- Wager, S., Hastie T., & Efron, B. (2014). Confidence Intervals for Random Forests: The Jackknife and the Infinitesimal Jackknife. J Mach Learn Res 15:1625-1651. https://jmlr.org/papers/v15/wager14a.html.
- Meinshausen (2006). Quantile Regression Forests. J Mach Learn Res 7:983-999. https://www.jmlr.org/papers/v7/meinshausen06a.html.

See Also

ranger

Examples

```
## Classification forest
ranger(Species ~ ., data = iris)
train.idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
rg.iris <- ranger(Species ~ ., data = iris.train)
pred.iris <- predict(rg.iris, data = iris.test)
table(iris.test$Species, pred.iris$predictions)

## Quantile regression forest
rf <- ranger(mpg ~ ., mtcars[1:26, ], quantreg = TRUE)
pred <- predict(rf, mtcars[27:32, ], type = "quantiles", quantiles = c(0.1, 0.5, 0.9))
pred$predictions

## Quantile regression forest with user-specified function
rf <- ranger(mpg ~ ., mtcars[1:26, ], quantreg = TRUE)
pred <- predict(rf, mtcars[27:32, ], type = "quantiles",
                what = function(x) sample(x, 10, replace = TRUE))
pred$predictions
```

---

predict.ranger.forest *Ranger prediction*

Description

Prediction with new data and a saved forest from Ranger.

Usage

```
## S3 method for class 'ranger.forest'
predict(
  object,
  data,
  predict.all = FALSE,
  num.trees = object$num.trees,
  type = "response",
  se.method = "infjack",
  seed = NULL,
  num.threads = NULL,
  verbose = TRUE,
  inbag.counts = NULL,
  ...
)
```

Arguments

object	Ranger ranger.forest object.
data	New test data of class data.frame or gwaa.data (GenABEL).
predict.all	Return individual predictions for each tree instead of aggregated predictions for all trees. Return a matrix (sample x tree) for classification and regression, a 3d array for probability estimation (sample x class x tree) and survival (sample x time x tree).
num.trees	Number of trees used for prediction. The first num.trees in the forest are used.
type	Type of prediction. One of 'response', 'se', 'terminalNodes', 'quantiles' with default 'response'. See below for details.
se.method	Method to compute standard errors. One of 'jack', 'infjack' with default 'infjack'. Only applicable if type = 'se'. See below for details.
seed	Random seed. Default is NULL, which generates the seed from R. Set to 0 to ignore the R seed. The seed is used in case of ties in classification mode.
num.threads	Number of threads. Use 0 for all available cores. Default is 2 if not set by options/environment variables (see below).
verbose	Verbose output on or off.
inbag.counts	Number of times the observations are in-bag in the trees.
...	further arguments passed to or from other methods.

Details

For type = 'response' (the default), the predicted classes (classification), predicted numeric values (regression), predicted probabilities (probability estimation) or survival probabilities (survival) are returned. For type = 'se', the standard error of the predictions are returned (regression only). The jackknife-after-bootstrap or infinitesimal jackknife for bagging is used to estimate the standard errors based on out-of-bag predictions. See Wager et al. (2014) for details. For type = 'terminalNodes', the IDs of the terminal node in each tree for each observation in the given dataset are returned.

If type = 'se' is selected, the method to estimate the variances can be chosen with se.method. Set se.method = 'jack' for jackknife after bootstrap and se.method = 'infjack' for the infinitesimal jackknife for bagging.

For classification and predict.all = TRUE, a factor levels are returned as numerics. To retrieve the corresponding factor levels, use rf$forest$levels, if rf is the ranger object.

By default, ranger uses 2 threads. The default can be changed with: (1) num.threads in ranger/predict call, (2) environment variable R_RANGER_NUM_THREADS, (3) options(ranger.num.threads = N), (4) options(Ncpus = N), with precedence in that order.

Value

Object of class ranger.prediction with elements

predictions	Predicted classes/values (only for classification and regression)
unique.death.times	Unique death times (only for survival).

chf	Estimated cumulative hazard function for each sample (only for survival).
survival	Estimated survival function for each sample (only for survival).
num.trees	Number of trees.
num.independent.variables	Number of independent variables.
treetype	Type of forest/tree. Classification, regression or survival.
num.samples	Number of samples.

Author(s)

Marvin N. Wright

References

- Wright, M. N. & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.
- Wager, S., Hastie T., & Efron, B. (2014). Confidence Intervals for Random Forests: The Jackknife and the Infinitesimal Jackknife. J Mach Learn Res 15:1625-1651. https://jmlr.org/papers/v15/wager14a.html.

See Also

ranger

---

predictions.ranger *Ranger predictions*

Description

Extract training data predictions of Ranger object.

Usage

```
## S3 method for class 'ranger'
predictions(x, ...)
```

Arguments

x	Ranger object.
...	Further arguments passed to or from other methods.

Value

Predictions: Classes for Classification forests, Numerical values for Regressions forests and the estimated survival functions for all individuals for Survival forests.

Author(s)

Marvin N. Wright

See Also

ranger

---

predictions.ranger.prediction
Ranger predictions

Description

Extract predictions of Ranger prediction object.

Usage

```
## S3 method for class 'ranger.prediction'
predictions(x, ...)
```

Arguments

x	Ranger prediction object.
...	Further arguments passed to or from other methods.

Value

Predictions: Classes for Classification forests, Numerical values for Regressions forests and the estimated survival functions for all individuals for Survival forests.

Author(s)

Marvin N. Wright

See Also

ranger

print.deforest.ranger *Print deforested ranger summary*

Description

Print basic information about a deforested `ranger` object.

Usage

```
## S3 method for class 'deforest.ranger'
print(x, ...)
```

Arguments

`x`	A `deforest` object (i.e., an object that inherits from class `"deforest.ranger"`).
`...`	Further arguments passed to or from other methods.

Note

Many of the components of a typical `ranger` object are not available after deforestation and are instead replaced with `NA` (e.g., out-of-bag (OOB) predictions, variable importance scores (if requested), and OOB-based error metrics).

Author(s)

Brandon M. Greenwell

See Also

`deforest`.

print.ranger *Print Ranger*

Description

Print contents of Ranger object.

Usage

```
## S3 method for class 'ranger'
print(x, ...)
```

Arguments

`x`	Object of class 'ranger'.
`...`	Further arguments passed to or from other methods.

Author(s)

Marvin N. Wright

See Also

ranger

---

print.ranger.forest *Print Ranger forest*

Description

Print contents of Ranger forest object.

Usage

```
## S3 method for class 'ranger.forest'
print(x, ...)
```

Arguments

x	Object of class 'ranger.forest'.
...	further arguments passed to or from other methods.

Author(s)

Marvin N. Wright

---

print.ranger.prediction *Print Ranger prediction*

Description

Print contents of Ranger prediction object.

Usage

```
## S3 method for class 'ranger.prediction'
print(x, ...)
```

Arguments

x	Object of class 'ranger.prediction'.
...	further arguments passed to or from other methods.

Author(s)

Marvin N. Wright

`ranger` *Ranger*

Description

Ranger is a fast implementation of random forests (Breiman 2001) or recursive partitioning, particularly suited for high dimensional data. Classification, regression, and survival forests are supported. Classification and regression forests are implemented as in the original Random Forest (Breiman 2001), survival forests as in Random Survival Forests (Ishwaran et al. 2008). Includes implementations of extremely randomized trees (Geurts et al. 2006) and quantile regression forests (Meinshausen 2006).

Usage

```
ranger(
  formula = NULL,
  data = NULL,
  num.trees = 500,
  mtry = NULL,
  importance = "none",
  write.forest = TRUE,
  probability = FALSE,
  min.node.size = NULL,
  min.bucket = NULL,
  max.depth = NULL,
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  case.weights = NULL,
  class.weights = NULL,
  splitrule = NULL,
  num.random.splits = 1,
  alpha = 0.5,
  minprop = 0.1,
  poisson.tau = 1,
  split.select.weights = NULL,
  always.split.variables = NULL,
  respect.unordered.factors = NULL,
  scale.permutation.importance = FALSE,
  local.importance = FALSE,
  regularization.factor = 1,
  regularization.usedepth = FALSE,
  keep.inbag = FALSE,
  inbag = NULL,
```

```
  holdout = FALSE,
  quantreg = FALSE,
  time.interest = NULL,
  oob.error = TRUE,
  num.threads = NULL,
  save.memory = FALSE,
  verbose = TRUE,
  node.stats = FALSE,
  seed = NULL,
  na.action = "na.learn",
  dependent.variable.name = NULL,
  status.variable.name = NULL,
  classification = NULL,
  x = NULL,
  y = NULL,
  ...
)
```

Arguments

formula	Object of class formula or character describing the model to fit. Interaction terms supported only for numerical variables.
data	Training data of class data.frame, matrix, dgCMatrix (Matrix) or gwaa.data (GenABEL).
num.trees	Number of trees.
mtry	Number of variables to possibly split at in each node. Default is the (rounded down) square root of the number variables. Alternatively, a single argument function returning an integer, given the number of independent variables.
importance	Variable importance mode, one of 'none', 'impurity', 'impurity_corrected', 'permutation'. The 'impurity' measure is the Gini index for classification, the variance of the responses for regression and the sum of test statistics (see splitrule) for survival.
write.forest	Save ranger.forest object, required for prediction. Set to FALSE to reduce memory usage if no prediction intended.
probability	Grow a probability forest as in Malley et al. (2012).
min.node.size	Minimal node size to split at. Default 1 for classification, 5 for regression, 3 for survival, and 10 for probability. For classification, this can be a vector of class-specific values.
min.bucket	Minimal terminal node size. No nodes smaller than this value can occur. Default 3 for survival and 1 for all other tree types. For classification, this can be a vector of class-specific values.
max.depth	Maximal tree depth. A value of NULL or 0 (the default) corresponds to unlimited depth, 1 to tree stumps (1 split per tree).
replace	Sample with replacement.

`sample.fraction`
: Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement. For classification, this can be a vector of class-specific values.

`case.weights`
: Weights for sampling of training observations. Observations with larger weights will be selected with higher probability in the bootstrap (or subsampled) samples for the trees.

`class.weights`
: Weights for the outcome classes (in order of the factor levels) in the splitting rule (cost sensitive learning). Classification and probability prediction only. For classification the weights are also applied in the majority vote in terminal nodes.

`splitrule`
: Splitting rule. For classification and probability estimation "gini", "extratrees" or "hellinger" with default "gini". For regression "variance", "extratrees", "maxstat", "beta" or "poisson" with default "variance". For survival "logrank", "extratrees", "C" or "maxstat" with default "logrank".

`num.random.splits`
: For "extratrees" splitrule.: Number of random splits to consider for each candidate splitting variable.

`alpha`
: For "maxstat" splitrule: Significance threshold to allow splitting.

`minprop`
: For "maxstat" splitrule: Lower quantile of covariate distribution to be considered for splitting.

`poisson.tau`
: For "poisson" splitrule: The coefficient of variation of the (expected) frequency is $1/\tau$. If a terminal node has only 0 responses, the estimate is set to $\alpha 0 + (1 - \alpha) mean(parent)$ with $\alpha = samples(child) mean(parent) / (\tau + samples(child) mean(parent))$.

`split.select.weights`
: Numeric vector with weights between 0 and 1, used to calculate the probability to select variables for splitting. Alternatively, a list of size num.trees, containing split select weight vectors for each tree can be used.

`always.split.variables`
: Character vector with variable names to be always selected in addition to the `mtry` variables tried for splitting.

`respect.unordered.factors`
: Handling of unordered factor covariates. One of 'ignore', 'order' and 'partition'. For the "extratrees" splitrule the default is "partition" for all other splitrules 'ignore'. Alternatively TRUE (='order') or FALSE ='ignore') can be used. See below for details.

`scale.permutation.importance`
: Scale permutation importance by standard error as in (Breiman 2001). Only applicable if permutation variable importance mode selected.

`local.importance`
: Calculate and return local importance values as in (Breiman 2001). Only applicable if `importance` is set to 'permutation'.

`regularization.factor`
: Regularization factor (gain penalization), either a vector of length p or one value for all variables.

`regularization.usedepth`
: Consider the depth in regularization.

keep.inbag — Save how often observations are in-bag in each tree.

inbag — Manually set observations per tree. List of size num.trees, containing inbag counts for each observation. Can be used for stratified sampling.

holdout — Hold-out mode. Hold-out all samples with case weight 0 and use these for variable importance and prediction error.

quantreg — Prepare quantile prediction as in quantile regression forests (Meinshausen 2006). Regression only. Set keep.inbag = TRUE to prepare out-of-bag quantile prediction.

time.interest — Time points of interest (survival only). Can be NULL (default, use all observed time points), a vector of time points or a single number to use as many time points (grid over observed time points).

oob.error — Compute OOB prediction error. Set to FALSE to save computation time, e.g. for large survival forests.

num.threads — Number of threads. Use 0 for all available cores. Default is 2 if not set by options/environment variables (see below).

save.memory — Use memory saving (but slower) splitting mode. No effect for survival and GWAS data. Warning: This option slows down the tree growing, use only if you encounter memory problems.

verbose — Show computation status and estimated runtime.

node.stats — Save node statistics. Set to TRUE to save prediction, number of observations and split statistics for each node.

seed — Random seed. Default is NULL, which generates the seed from R. Set to 0 to ignore the R seed.

na.action — Handling of missing values. Set to "na.learn" to internally handle missing values (default, see below), to "na.omit" to omit observations with missing values and to "na.fail" to stop if missing values are found.

dependent.variable.name — Name of dependent variable, needed if no formula given. For survival forests this is the time variable.

status.variable.name — Name of status variable, only applicable to survival data and needed if no formula given. Use 1 for event and 0 for censoring.

classification — Set to TRUE to grow a classification forest. Only needed if the data is a matrix or the response numeric.

x — Predictor data (independent variables), alternative interface to data with formula or dependent.variable.name.

y — Response vector (dependent variable), alternative interface to data with formula or dependent.variable.name. For survival use a Surv() object or a matrix with time and status.

... — Further arguments passed to or from other methods (currently ignored).

Details

The tree type is determined by the type of the dependent variable. For factors classification trees are grown, for numeric values regression trees and for survival objects survival trees. The Gini index is used as default splitting rule for classification. For regression, the estimated response variances or maximally selected rank statistics (Wright et al. 2016) can be used. For Survival the log-rank test, a C-index based splitting rule (Schmid et al. 2015) and maximally selected rank statistics (Wright et al. 2016) are available. For all tree types, forests of extremely randomized trees (Geurts et al. 2006) can be grown.

With the `probability` option and factor dependent variable a probability forest is grown. Here, the node impurity is used for splitting, as in classification forests. Predictions are class probabilities for each sample. In contrast to other implementations, each tree returns a probability estimate and these estimates are averaged for the forest probability estimate. For details see Malley et al. (2012).

Note that nodes with size smaller than `min.node.size` can occur because `min.node.size` is the minimal node size *to split at*, as in original Random Forests. To restrict the size of terminal nodes, set `min.bucket`. Variables selected with `always.split.variables` are tried additionally to the mtry variables randomly selected. In `split.select.weights`, weights do not need to sum up to 1, they will be normalized later. The weights are assigned to the variables in the order they appear in the formula or in the data if no formula is used. Names of the `split.select.weights` vector are ignored. Weights assigned by `split.select.weights` to variables in `always.split.variables` are ignored. The usage of `split.select.weights` can increase the computation times for large forests.

Unordered factor covariates can be handled in 3 different ways by using `respect.unordered.factors`: For 'ignore' all factors are regarded ordered, for 'partition' all possible 2-partitions are considered for splitting. For 'order' and 2-class classification the factor levels are ordered by their proportion falling in the second class, for regression by their mean response, as described in Hastie et al. (2009), chapter 9.2.4. For multiclass classification the factor levels are ordered by the first principal component of the weighted covariance matrix of the contingency table (Coppersmith et al. 1999), for survival by the median survival (or the largest available quantile if the median is not available). The use of 'order' is recommended, as it computationally fast and can handle an unlimited number of factor levels. Note that the factors are only reordered once and not again in each split.

The 'impurity_corrected' importance measure is unbiased in terms of the number of categories and category frequencies and is almost as fast as the standard impurity importance. It is a modified version of the method by Sandri & Zuccolotto (2008), which is faster and more memory efficient. See Nembrini et al. (2018) for details. This importance measure can be combined with the methods to estimate p-values in `importance_pvalues`. We recommend not to use the 'impurity_corrected' importance when making predictions since the feature permutation step might reduce predictive performance (a warning is raised when predicting on new data).

Note that ranger has different default values than other packages. For example, our default for `mtry` is the square root of the number of variables for all tree types, whereas other packages use different values for regression. Also, changing one hyperparameter does not change other hyperparameters (where possible). For example, `splitrule="extratrees"` uses randomized splitting but does not disable bagging as in Geurts et al. (2006). To disable bagging, use `replace = FALSE, sample.fraction = 1`. This can also be used to grow a single decision tree without bagging and feature subsetting: `ranger(..., num.trees = 1, mtry = p, replace = FALSE, sample.fraction = 1)`, where p is the number of independent variables.

While random forests are known for their robustness, default hyperparameters not always work

well. For example, for high dimensional data, increasing the `mtry` value and the number of trees `num.trees` is recommended. For more details and recommendations, see Probst et al. (2019). To find the best hyperparameters, consider hyperparameter tuning with the `tuneRanger` or `mlr3` packages.

Out-of-bag prediction error is calculated as accuracy (proportion of misclassified observations) for classification, as Brier score for probability estimation, as mean squared error (MSE) for regression and as one minus Harrell's C-index for survival. Harrell's C-index is calculated based on the sum of the cumulative hazard function (CHF) over all timepoints, i.e., `rowSums(chf)`, where `chf` is the the out-of-bag CHF; for details, see Ishwaran et al. (2008). Calculation of the out-of-bag prediction error can be turned off with `oob.error = FALSE`.

Regularization works by penalizing new variables by multiplying the splitting criterion by a factor, see Deng & Runger (2012) for details. If `regularization.usedepth=TRUE`, f^d is used, where f is the regularization factor and d the depth of the node. If regularization is used, multithreading is deactivated because all trees need access to the list of variables that are already included in the model.

Missing values can be internally handled by setting `na.action = "na.learn"` (default), by omitting observations with missing values with `na.action = "na.omit"` or by stopping if missing values are found with `na.action = "na.fail"`. With `na.action = "na.learn"`, missing values are ignored for calculating an initial split criterion value (i.e., decrease of impurity). Then for the best split, all missings are tried in both child nodes and the choice is made based again on the split criterion value. For prediction, this direction is saved as the "default" direction. If a missing occurs in prediction at a node where there is no default direction, it goes left.

For a large number of variables and data frames as input data the formula interface can be slow or impossible to use. Alternatively `dependent.variable.name` (and `status.variable.name` for survival) or `x` and `y` can be used. Use `x` and `y` with a matrix for `x` to avoid conversions and save memory. Consider setting `save.memory = TRUE` if you encounter memory problems for very large datasets, but be aware that this option slows down the tree growing.

For GWAS data consider combining `ranger` with the `GenABEL` package. See the Examples section below for a demonstration using `Plink` data. All SNPs in the `GenABEL` object will be used for splitting. To use only the SNPs without sex or other covariates from the phenotype file, use `0` on the right hand side of the formula. Note that missing values are treated as an extra category while splitting.

By default, ranger uses 2 threads. The default can be changed with: (1) `num.threads` in ranger/predict call, (2) environment variable R_RANGER_NUM_THREADS, (3) `options(ranger.num.threads = N)`, (4) `options(Ncpus = N)`, with precedence in that order.

See https://github.com/imbs-hl/ranger for the development version.

Value

Object of class `ranger` with elements

`forest` Saved forest (If write.forest set to TRUE). Note that the variable IDs in the `split.varIDs` object do not necessarily represent the column number in R.

`predictions` Predicted classes/values, based on out-of-bag samples (classification and regression only).

`variable.importance`
Variable importance for each independent variable.

`variable.importance.local`
: Variable importance for each independent variable and each sample, if `local.importance` is set to TRUE and `importance` is set to 'permutation'.

`prediction.error`
: Overall out-of-bag prediction error. For classification this is accuracy (proportion of misclassified observations), for probability estimation the Brier score, for regression the mean squared error and for survival one minus Harrell's C-index.

`r.squared`
: R squared. Also called explained variance or coefficient of determination (regression only). Computed on out-of-bag data.

`confusion.matrix`
: Contingency table for classes and predictions based on out-of-bag samples (classification only).

`unique.death.times`
: Unique death times (survival only).

`chf`
: Estimated cumulative hazard function for each sample (survival only).

`survival`
: Estimated survival function for each sample (survival only).

`call`
: Function call.

`num.trees`
: Number of trees.

`num.independent.variables`
: Number of independent variables.

`mtry`
: Value of mtry used.

`min.node.size`
: Value of minimal node size used.

`treetype`
: Type of forest/tree. classification, regression or survival.

`importance.mode`
: Importance mode used.

`num.samples`
: Number of samples.

`inbag.counts`
: Number of times the observations are in-bag in the trees.

`dependent.variable.name`
: Name of the dependent variable. This is NULL when x/y interface is used.

`status.variable.name`
: Name of the status variable (survival only). This is NULL when x/y interface is used.

Author(s)

Marvin N. Wright

References

- Wright, M. N. & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1-17. doi:10.18637/jss.v077.i01.
- Schmid, M., Wright, M. N. & Ziegler, A. (2016). On the use of Harrell's C for clinical risk prediction via random survival forests. Expert Syst Appl 63:450-459. doi:10.1016/j.eswa.2016.07.018.

- Wright, M. N., Dankowski, T. & Ziegler, A. (2017). Unbiased split variable selection for random survival forests using maximally selected rank statistics. Stat Med 36:1272-1284. doi:10.1002/sim.7212.
- Nembrini, S., Koenig, I. R. & Wright, M. N. (2018). The revival of the Gini Importance? Bioinformatics. doi:10.1093/bioinformatics/bty373.
- Breiman, L. (2001). Random forests. Mach Learn, 45:5-32. doi:10.1023/A:1010933404324.
- Ishwaran, H., Kogalur, U. B., Blackstone, E. H., & Lauer, M. S. (2008). Random survival forests. Ann Appl Stat 2:841-860. doi:10.1097/JTO.0b013e318233d835.
- Malley, J. D., Kruppa, J., Dasgupta, A., Malley, K. G., & Ziegler, A. (2012). Probability machines: consistent probability estimation using nonparametric learning machines. Methods Inf Med 51:74-81. doi:10.3414/ME00010052.
- Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning. Springer, New York. 2nd edition.
- Geurts, P., Ernst, D., Wehenkel, L. (2006). Extremely randomized trees. Mach Learn 63:3-42. doi:10.1007/s1099400662261.
- Meinshausen (2006). Quantile Regression Forests. J Mach Learn Res 7:983-999. https://www.jmlr.org/papers/v7/meinshausen06a.html.
- Sandri, M. & Zuccolotto, P. (2008). A bias correction algorithm for the Gini variable importance measure in classification trees. J Comput Graph Stat, 17:611-628. doi:10.1198/106186008X344522.
- Coppersmith D., Hong S. J., Hosking J. R. (1999). Partitioning nominal attributes in decision trees. Data Min Knowl Discov 3:197-217. doi:10.1023/A:1009869804967.
- Deng & Runger (2012). Feature selection via regularized trees. The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia. doi:10.1109/IJCNN.2012.6252640.
- Probst, P., Wright, M. N. & Boulesteix, A-L. (2019). Hyperparameters and tuning strategies for random forest. WIREs Data Mining Knowl Discov 9:e1301.doi:10.1002/widm.1301.

See Also

predict.ranger

Examples

```
## Classification forest with default settings
ranger(Species ~ ., data = iris)

## Prediction
train.idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
rg.iris <- ranger(Species ~ ., data = iris.train)
pred.iris <- predict(rg.iris, data = iris.test)
table(iris.test$Species, pred.iris$predictions)

## Quantile regression forest
rf <- ranger(mpg ~ ., mtcars[1:26, ], quantreg = TRUE)
pred <- predict(rf, mtcars[27:32, ], type = "quantiles")
```

```
pred$predictions

## Variable importance
rg.iris <- ranger(Species ~ ., data = iris, importance = "impurity")
rg.iris$variable.importance

## Survival forest
require(survival)
rg.veteran <- ranger(Surv(time, status) ~ ., data = veteran)
plot(rg.veteran$unique.death.times, rg.veteran$survival[1,])

## Alternative interfaces (same results)
ranger(dependent.variable.name = "Species", data = iris)
ranger(y = iris[, 5], x = iris[, -5])

## Not run:
## Use GenABEL interface to read Plink data into R and grow a classification forest
## The ped and map files are not included
library(GenABEL)
convert.snp.ped("data.ped", "data.map", "data.raw")
dat.gwaa <- load.gwaa.data("data.pheno", "data.raw")
phdata(dat.gwaa)$trait <- factor(phdata(dat.gwaa)$trait)
ranger(trait ~ ., data = dat.gwaa)

## End(Not run)
```

`timepoints.ranger` *Ranger timepoints*

Description

Extract unique death times of Ranger Survival forest

Usage

```
## S3 method for class 'ranger'
timepoints(x, ...)
```

Arguments

`x`	Ranger Survival forest object.
`...`	Further arguments passed to or from other methods.

Value

Unique death times

Author(s)

Marvin N. Wright

See Also

ranger

`timepoints.ranger.prediction` *Ranger timepoints*

Description

Extract unique death times of Ranger Survival prediction object.

Usage

```
## S3 method for class 'ranger.prediction'
timepoints(x, ...)
```

Arguments

x	Ranger Survival prediction object.
...	Further arguments passed to or from other methods.

Value

Unique death times

Author(s)

Marvin N. Wright

See Also

ranger

`treeInfo` *Tree information in human readable format*

Description

Extract tree information of a `ranger` object.

Usage

```
treeInfo(object, tree = 1)
```

Arguments

`object`	`ranger` object.
`tree`	Number of the tree of interest.

Details

Node and variable ID's are 0-indexed, i.e., node 0 is the root node. If the formula interface is used in the `ranger` call, the variable ID's are usually different to the original data used to grow the tree. Refer to the variable name instead to be sure.

Splitting at unordered factors (nominal variables) depends on the option `respect.unordered.factors` in the `ranger` call. For the "ignore" and "order" approaches, all values smaller or equal the `splitval` value go to the left and all values larger go to the right, as usual. However, with "order" the values correspond to the order in `object$forest$covariate.levels` instead of the original order (usually alphabetical). In the "partition" mode, the `splitval` values for unordered factor are comma separated lists of values, representing the factor levels (in the original order) going to the right.

Value

A data.frame with the columns

`nodeID`	The nodeID, 0-indexed.
`leftChild`	ID of the left child node, 0-indexed.
`rightChild`	ID of the right child node, 0-indexed.
`splitvarID`	ID of the splitting variable, 0-indexed. Caution, the variable order changes if the formula interface is used.
`splitvarName`	Name of the splitting variable.
`splitval`	The splitting value. For numeric or ordinal variables, all values smaller or equal go to the left, larger values t
`terminal`	Logical, TRUE for terminal nodes.
`prediction`	One column with the predicted class (factor) for classification and the predicted numerical value for regressi
`numSamples`	Number of samples in the node (only if ranger called with `node.stats = TRUE`).
`splitStat`	Split statistics, i.e., value of the splitting criterion (only if ranger called with `node.stats = TRUE`).

Author(s)

Marvin N. Wright

See Also

ranger

Examples

```
rf <- ranger(Species ~ ., data = iris)
treeInfo(rf, 1)
```

Index