Package ‘diversityForest’

March 8, 2023

Type Package

Title Innovative Complex Split Procedures in Random Forests Through Candidate Split Sampling

Version 0.4.0

Date 2023-03-07

Author Roman Hornung [aut, cre], Marvin N. Wright [ctb, cph]

Maintainer Roman Hornung <hornung@ibe.med.uni-muenchen.de>

Description Implements interaction forests [1], which are specific diversity forests and the basic form of diversity forests that uses univariable, binary splitting [2].
Interaction forests (IFs) are ensembles of decision trees that model quantitative and qualitative interaction effects using bivariable splitting. IFs come with the Effect Importance Measure (EIM), which can be used to identify variable pairs that feature quantitative and qualitative interaction effects with high predictive relevance. IFs and EIM focus on well interpretable forms of interactions.
The package also offers plot functions for visualising the estimated forms of interaction effects.
Categorical, metric, and survival outcomes are supported.
This is a fork of the R package 'ranger' (main author: Marvin N. Wright) that implements random forests using an efficient C++ implementation.
References:
[1] Hornung, R. & Boulesteix, A.-L. (2022) Interaction Forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects.
Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
[2] Hornung, R. (2022) Diversity forests: Using split sampling to enable innovative complex split procedures in random forests.
SN Computer Science 3(2):1, <doi:10.1007/s42979-021-00920-1>.

SystemRequirements C++17

Encoding UTF-8

License GPL-3

Imports Rcpp (>= 0.11.2), Matrix, ggplot2, ggpubr, scales, nnet, sgeostat, rms, MapGAM, gam, rlang, grDevices, RColorBrewer, RcppEigen, survival

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.5)

Suggests testthat, BOLTSSIRR

Additional_repositories `https://romanhornung.github.io/drat`

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-03-08 08:20:02 UTC

R topics documented:

`diversityForest-package` *Diversity Forests*

Description

The diversity forest algorithm is not a specific algorithm, but an alternative candidate split sampling scheme that makes complex split procedures in random forests possible computationally by drastically reducing the numbers of candidate splits that need to be evaluated for each split. It also avoids the well-known variable selection bias in conventional random forests that has the effect that variables with many possible splits are selected too frequently for splitting (Strobl et al., 2007). For details, see Hornung (2022).

Details

This package currently features two types of diversity forests:

- the basic form of diversity forests that uses univariable, binary splitting, which is also used in conventional random forests

- interaction forests (IFs) (Hornung & Boulesteix, 2022), which use bivariable splitting to model quantitative and qualitative interaction effects. IFs feature the Effect Importance Measure (EIM), which ranks the variable pairs with respect to the predictive importance of their quantitative and qualitative interaction effects. The individual variables can be ranked as well using EIM. For details, see Hornung & Boulesteix (2022).

Diversity forests with univariable splitting can be constructed using the function `divfor` and interaction forests using the function `interactionfor`. Both functions support categorical, metric, and survival outcomes.

This package is a fork of the R package 'ranger' that implements random forests using an efficient C++ implementation. The documentation is in large parts taken from 'ranger', where some parts of the documentation may not apply to (the current version of) the 'diversityForest' package.

Details on further functionalities of the code that are not presented in the help pages of 'diversityForest' are found in the help pages of 'ranger', version 0.11.0, because 'diversityForest' is based on the latter version of 'ranger'. The code in the example sections can be used as a template for all basic application scenarios with respect to classification, regression and survival prediction.

References

- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
- Strobl, C., Boulesteix, A.-L., Zeileis, A., Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8:25, <doi:10.1186/14712105825>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 77:1-17, <doi:10.18637/jss.v077.i01>.

`divfor` *Construct a basic diversity forest prediction rule that uses univariable, binary splitting.*

Description

Implements the most basic form of diversity forests that uses univariable, binary splitting. Currently, categorical, metric, and survival outcomes are supported.

Usage

```
divfor(
  formula = NULL,
  data = NULL,
  num.trees = 500,
  mtry = NULL,
  importance = "none",
  write.forest = TRUE,
  probability = FALSE,
  min.node.size = 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,
  split.select.weights = NULL,
  always.split.variables = NULL,
  respect.unordered.factors = NULL,
  scale.permutation.importance = FALSE,
  keep.inbag = FALSE,
  inbag = NULL,
  holdout = FALSE,
  quantreg = FALSE,
  oob.error = TRUE,
  num.threads = NULL,
  save.memory = FALSE,
  verbose = TRUE,
  seed = NULL,
  dependent.variable.name = NULL,
  status.variable.name = NULL,
  classification = NULL,
  nsplits = 30,
  proptry = 1
)
```

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. Default is 500.
`mtry`	Artefact from 'ranger'. NOT needed for diversity forests.

`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. NOTE: Currently, only "permutation" (and "none") work for diversity forests.
`write.forest`	Save `divfor.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). NOTE: Not yet implemented for diversity forests!
`min.node.size`	Minimal node size. Default 1 for classification, 5 for regression, 3 for survival, and 5 for probability.
`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" or "extratrees" with default "gini". For regression "variance", "extratrees" or "maxstat" with default "variance". For survival "logrank", "extratrees", "C" or "maxstat" with default "logrank". NOTE: For diversity forests currently only the default splitting rules are supported.
`num.random.splits`	Artefact from 'ranger'. NOT needed for diversity forests.
`alpha`	For "maxstat" splitrule: Significance threshold to allow splitting. NOT needed for diversity forests.
`minprop`	For "maxstat" splitrule: Lower quantile of covariate distribution to be considered for splitting. NOT needed for diversity forests.
`split.select.weights`	Numeric vector with weights between 0 and 1, representing 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`	Currently not useable. Character vector with variable names to be always selected.
`respect.unordered.factors`	Handling of unordered factor covariates. One of 'ignore' and 'order' (the option 'partition' possible in 'ranger' is not (yet) possible with diversity forests). Default is 'ignore'. Alternatively TRUE (='order') or FALSE (='ignore') can be used.

`scale.permutation.importance`	Scale permutation importance by standard error as in (Breiman 2001). Only applicable if permutation variable importance mode selected.
`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.
`oob.error`	Compute OOB prediction error. Set to `FALSE` to save computation time, e.g. for large survival forests.
`num.threads`	Number of threads. Default is number of CPUs available.
`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. NOT needed for diversity forests.
`verbose`	Show computation status and estimated runtime.
`seed`	Random seed. Default is `NULL`, which generates the seed from `R`. Set to `0` to ignore the `R` seed.
`dependent.variable.name`	Name of outcome 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`	Only needed if data is a matrix. Set to `TRUE` to grow a classification forest.
`nsplits`	Number of candidate splits to sample for each split. Default is 30.
`proptry`	Parameter that restricts the number of candidate splits considered for small nodes. If `nsplits` is larger than `proptry` times the number of all possible splits, the number of candidate splits to draw is reduced to the largest integer smaller than `proptry` times the number of all possible splits. Default is 1, which corresponds to always using `nsplits` candidate splits.

Value

Object of class `divfor` 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.

`prediction.error`	Overall out-of-bag prediction error. For classification this is the fraction of missclassified samples, 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.
`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.
`splitrule`	Splitting rule.
`replace`	Sample with replacement.
`nsplits`	Value of `nsplits` used.
`proptry`	Value of `proptry` used.

Author(s)

Roman Hornung, Marvin N. Wright

References

- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. Journal of Statistical Software 77:1-17, <doi:10.18637/jss.v077.i01>.
- Breiman, L. (2001). Random forests. Machine Learning 45:5-32, <doi:10.1023/A:1010933404324>.
- Malley, J. D., Kruppa, J., Dasgupta, A., Malley, K. G., & Ziegler, A. (2012). Probability machines: consistent probability estimation using nonparametric learning machines. Methods of Information in Medicine 51:74-81, <doi:10.3414/ME00010052>.
- Meinshausen (2006). Quantile Regression Forests. Journal of Machine Learning Research 7:983-999.

See Also

predict.divfor

Examples

```
## Not run:

## Load package:
library("diversityForest")

## Set seed to obtain reproducible results:
set.seed(1234)

## Diversity forest with default settings (NOT recommended)
# Classification:
divfor(Species ~ ., data = iris, num.trees = 20)
# Regression:
iris2 <- iris; iris2$Species <- NULL; iris2$Y <- rnorm(nrow(iris2))
divfor(Y ~ ., data = iris2, num.trees = 20)
# Survival:
library("survival")
divfor(Surv(time, status) ~ ., data = veteran, num.trees = 20, respect.unordered.factors = "order")
# NOTE: num.trees = 20 is specified too small for practical
# purposes - the prediction performance of the resulting
# forest will be suboptimal!!
# In practice, num.trees = 500 (default value) or a
# larger number should be used.

## Diversity forest with specified values for nsplits and proptry (NOT recommended)
divfor(Species ~ ., data = iris, nsplits = 10, proptry = 0.4, num.trees = 20)
# NOTE again: num.trees = 20 is specified too small for practical purposes.

## Applying diversity forest after optimizing the values of nsplits and proptry (recommended)
tuneres <- tunedivfor(formula = Species ~ ., data = iris, num.trees.pre = 20)
# NOTE: num.trees.pre = 20 is specified too small for practical
# purposes - the out-of-bag error estimates of the forests
# constructed during optimization will be much too variable!!
# In practice, num.trees.pre = 500 (default value) or a
# larger number should be used.
divfor(Species ~ ., data = iris, nsplits = tuneres$nsplitsopt,
  proptry = tuneres$proptryopt, num.trees = 20)
# NOTE again: num.trees = 20 is specified too small for practical purposes.

## Prediction
train.idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
tuneres <- tunedivfor(formula = Species ~ ., data = iris.train, num.trees.pre = 20)
# NOTE again: num.trees.pre = 20 is specified too small for practical purposes.
rg.iris <- divfor(Species ~ ., data = iris.train, nsplits = tuneres$nsplitsopt,
  proptry = tuneres$proptryopt, num.trees = 20)
# NOTE again: num.trees = 20 is specified too small for practical purposes.
```

```
pred.iris <- predict(rg.iris, data = iris.test)
table(iris.test$Species, pred.iris$predictions)

## Variable importance
rg.iris <- divfor(Species ~ ., data = iris, importance = "permutation", num.trees = 20)
# NOTE again: num.trees = 20 is specified too small for practical purposes.
rg.iris$variable.importance

## End(Not run)
```

`importance.divfor` *Diversity Forest variable importance*

Description

Extract variable importance of `divfor` object.

Usage

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

Arguments

`x`	`divfor` object.
`...`	Further arguments passed to or from other methods.

Value

Variable importance measures.

Author(s)

Marvin N. Wright

See Also

`divfor`

`interactionfor` *Construct an interaction forest prediction rule and calculate EIM values as described in Hornung & Boulesteix (2022).*

Description

Implements interaction forests as described in Hornung & Boulesteix (2022). Currently, categorical, metric, and survival outcomes are supported. Interaction forests feature the effect importance measure (EIM), which can be used to rank the covariate variable pairs with respect to the impact of their interaction effects on prediction. This allows to identify relevant interaction effects. Interaction forests focus on well interpretable interaction effects. See the 'Details' section below for more details. In addition, we strongly recommend to consult Section C of Supplementary Material 1 of Hornung & Boulesteix (2022), which uses detailed examples of interaction forest analyses with code to illustrate how interaction forests can be used in applications: Link.

Usage

```
interactionfor(
  formula = NULL,
  data = NULL,
  importance = "both",
  num.trees = NULL,
  simplify.large.n = TRUE,
  num.trees.eim.large.n = NULL,
  write.forest = TRUE,
  probability = FALSE,
  min.node.size = NULL,
  max.depth = NULL,
  replace = FALSE,
  sample.fraction = ifelse(replace, 1, 0.7),
  case.weights = NULL,
  class.weights = NULL,
  splitrule = NULL,
  always.split.variables = NULL,
  keep.inbag = FALSE,
  inbag = NULL,
  holdout = FALSE,
  quantreg = FALSE,
  oob.error = TRUE,
  num.threads = NULL,
  verbose = TRUE,
  seed = NULL,
  dependent.variable.name = NULL,
  status.variable.name = NULL,
  npairs = NULL,
  classification = NULL
)
```

Arguments

`formula`	Object of class `formula` or `character` describing the model to fit.
`data`	Training data of class `data.frame`, `matrix`, `dgCMatrix` (Matrix) or `gwaa.data` (GenABEL).
`importance`	Effect importance mode. One of the following: "both" (the default), "qualitative", "quantitative", "mainonly", "none". See the 'Details' section below for explanation.
`num.trees`	Number of trees. The default number is 20000, if EIM values should be computed and 2000 otherwise. Note that if `simplify.large.n = TRUE` (default), the number of observations is larger than 1000, and EIM values should be calculated two forests are constructed, one for calculating the EIM values and one for prediction (cf. 'Details' section). In such cases, the default number of trees used for the forest for EIM value calculation is 20000 and the default number of trees used for the forest for prediction is 2000.
`simplify.large.n`	Should restricted tree depths be used, when calculating EIM values for large data sets? See the 'Details' section below for more information. Default is `TRUE`.
`num.trees.eim.large.n`	Number of trees in the forest used for calculating the EIM values for large data sets. If `num.trees` is provided, but not `num.trees.eim.large.n`, the value given by `num.trees` will be used. The default number is 20000. Only used when `simplify.large.n = TRUE`.
`write.forest`	Save `interaction.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. Default 1 for classification, 5 for regression, 3 for survival, and 5 for probability.
`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. Default is `FALSE`.
`sample.fraction`	Fraction of observations to sample. Default is 1 for sampling with replacement and 0.7 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" or "extratrees" with default "gini". For regression "variance", "extratrees" or "maxstat" with default "variance". For survival "logrank", "extratrees", "C" or "maxstat" with default "logrank". NOTE: For interaction forests currently only the default splitting rules are supported.

`always.split.variables`	Currently not useable. Character vector with variable names to be always selected.
`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. NOTE: Currently, not useable for interaction forests.
`quantreg`	Prepare quantile prediction as in quantile regression forests (Meinshausen 2006). Regression only. Set `keep.inbag = TRUE` to prepare out-of-bag quantile prediction. NOTE: Currently, not useable for interaction forests.
`oob.error`	Compute OOB prediction error. Set to `FALSE` to save computation time, e.g. for large survival forests.
`num.threads`	Number of threads. Default is number of CPUs available.
`verbose`	Show computation status and estimated runtime.
`seed`	Random seed. Default is `NULL`, which generates the seed from `R`. Set to `0` to ignore the `R` seed.
`dependent.variable.name`	Name of outcome 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.
`npairs`	Number of variable pairs to sample for each split. Default is the square root of the number of independent variables divided by 2 (this number is rounded up).
`classification`	Only needed if data is a matrix. Set to `TRUE` to grow a classification forest.

Details

The effect importance measure (EIM) of interaction forests distinguishes quantitative and qualitative interaction effects (Peto, 1982). This is a common distinction as these two types of interaction effects are interpreted in different ways (see below). For both of these types, EIM values for each variable pair are obtained: the quantitative and qualitative EIM values.

Interaction forests target easily interpretable types of interaction effects. These can be communicated clearly using statements of the following kind: "The strength of the positive (negative) effect of variable A on the outcome depends on the level of variable B" for quantitative interactions, and "for observations with small values of variable B, the effect of variable A is positive (negative), but for observations with large values of B, the effect of A is negative (positive)" for qualitative interactions.

In addition to calculating EIM values for variable pairs, importance values for the individual variables are calculated as well, the univariable EIM values. These measure the variable importance as in the case of classical variable importance measures of random forests.

The effect importance mode can be set via the `importance` argument: `"qualitative"`: Calculate only qualitative EIM values; `"quantitative"`: Calculate only quantitative EIM values; `"both"`

(the default): Calculate qualitative and quantitative EIM values; `"mainonly"`: Calculate only univariable EIM values.
The top variable pairs with largest quantitative and qualitative EIM values likely have quantitative and qualitative interactions, respectively, which have a considerable impact on prediction. The top variables with largest univariable EIM values likely have a considerable impact on prediction. Note that it is currently not possible to test the EIM values for statistical significance using the interaction forests algorithm itself. However, the p-values shown in the plots obtained with `plotEffects` (which are obtained using bivariable models) can be adjusted for multiple testing using the Bonferroni procedure to obtain practical p-values. See the end of the 'Details' section of `plotEffects` for explanation and guidance.
If the number of variables is larger than 100, not all possible variable pairs are considered, but, using a screening procedure, the 5000 variable pairs with the strongest indications of interaction effects are pre-selected.
NOTE: To make interpretations, it is crucial to investigate (visually) the forms the interaction effects of variable pairs with large quantitative and qualitative EIM values take. This can be done using the plot function `plot.interactionfor` (first overview) and `plotEffects`.
NOTE ALSO: As described in Hornung & Boulesteix (2022), in the case of data with larger numbers of variables (larger than 100, but more seriously for high-dimensional data), the univariable EIM values can be biased. Therefore, it is strongly recommended to interpret the univariable EIM values with caution, if the data are high-dimensional. If it is of interest to measure the univariable importance of the variables for high-dimensional data, an additional conventional random forest (e.g., using the `ranger` package) should be constructed and the variable importance measure values of this random forest be used for ranking the univariable effects.
For large data sets with many observations the calculation of the EIM values can become very costly - when using fully grown trees. Therefore, when calculating EIM values for data sets with more than 1000 observations we use the following maximum tree depths by default (argument: `simplify.large.n = TRUE`):

- if $n \leq 1000$: Use fully grown trees.
- if $1000 < n \leq 2000$: Use tree depth 10.
- if $2000 < n \leq 5000$: Use tree depth 7.
- if $n > 5000$: Use tree depth 5.

Extensive analyses in Hornung & Boulesteix (2022) suggest that by restricting the tree depth in this way, the EIM values that would result when using fully grown trees are approximated well. However, the prediction performance suffers, when using restricted trees. Therefore, we restrict the tree depth only when calculating the EIM values (if $n > 1000$), but construct a second interaction forest with unrestricted tree depth, which is then used for prediction purposes.

Value

Object of class `interactionfor` with elements

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

`unique.death.times`	Unique death times (survival only).
`min.node.size`	Value of minimal node size used.
`npairs`	Number of variable pairs sampled for each split.
`eim.univ.sorted`	Univariable EIM values sorted in decreasing order.
`eim.univ`	Univariable EIM values.
`eim.qual.sorted`	Qualitative EIM values sorted in decreasing order.
`eim.qual`	Qualitative EIM values.
`eim.quant.sorted`	Quantitative EIM values sorted in decreasing order. The labeling of these values provides the information on the type of quantitative interactions the respective variable pairs feature. For example, consider a variable pair A and B and say the label reads "A large AND B small". This would mean that if the value of A is large and, at the same time, the value of B is small, the expected value of the outcome variable is (considerably) different from all other cases. For this type of quantitative interaction, the effect of B is weak for small values of A and strong for large values of A. See Hornung & Boulesteix (2022) for more information on the types of quantitative interaction effects targeted by interaction forest.
`eim.quant`	Quantitative EIM values. These values are labeled analoguously as those in `eim.quant.sorted`.
`prediction.error`	Overall out-of-bag prediction error. For classification this is the fraction of misclassified samples, for probability estimation the Brier score, for regression the mean squared error and for survival one minus Harrell's C-index. This is 'NA' for data sets with more than 100 covariate variables, because for such data sets we pre-select the 5000 variable pairs with strongest indications of interaction effects. This pre-selection cannot be taken into account in the out-of-bag error estimation, which is why the out-of-bag error estimates would be (much) too optimistic for data sets with more than 100 covariate variables.
`forest`	Saved forest (If write.forest set to TRUE). Note that the variable IDs in the `split.multvarIDs` object do not necessarily represent the column number in R.
`confusion.matrix`	Contingency table for classes and predictions based on out-of-bag samples (classification only).
`chf`	Estimated cumulative hazard function for each sample (survival only).
`survival`	Estimated survival function for each sample (survival only).
`splitrule`	Splitting rule.
`treetype`	Type of forest/tree. classification, regression or survival.
`r.squared`	R squared. Also called explained variance or coefficient of determination (regression only). Computed on out-of-bag data.
`call`	Function call.

`importance.mode`	Importance mode used.
`num.samples`	Number of samples.
`replace`	Sample with replacement.
`eim.quant.rawlists`	List containing the four vectors of un-adjusted 'raw' quantitative EIM values and the four vectors of adjusted EIM values. These are usually not required by the user. For each of the four types of quantitative splits there exists a separate vector of raw quantitative EIM values. For example, `eim.quant.large.small.raw` contains the raw quantitative EIM values of the quantitative split type associated with quantitative interaction effects for which the expected values of the outcome variable are different, if the value of variable A is large and, at the same time, the value of variable B is small. The list entries of the un-adjusted 'raw' quantitative EIM values are labeled with the suffix `.raw`, while the list entries of the adjusted quantitative EIM values miss this suffix. See Hornung & Boulesteix (2022) for details on the raw and adjusted EIM values.
`promispairs`	List giving the indices of the variables in the pre-selected variable pairs. If the number of variables is at most 100, all variable pairs are considered.
`plotres`	List ob objects needed by the plot functions: `eim.univ.order` contains the sorting of the univariable EIM values in descending order, where the first element gives the index of the variable with largest EIM value, the second element the index of the variable with second-largest EIM value and so on; `eim.qual.order` / `eim.quant.order` contains the sorting in descending order of the qualitative / quantitative EIM values for the (pre-selected) variable pairs given by the object `promispairs` above. The first element gives the index of the (pre-selected) variable pair with largest qualitative / quantitative EIM value, the second element the index of the variable pair with second-largest qualitative / quantitative EIM value; `data` contains the data; `yvarname` is the name of the outcome variable (survival time for survival); `statusvarname` is the name of the status variable.

Author(s)

Roman Hornung, Marvin N. Wright

References

- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Peto, R., (1982) Statistical aspects of cancer trials. In: K.E. Halnam (Ed.), Treatment of Cancer. Chapman & Hall: London.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 77:1-17, <doi:10.18637/jss.v077.i01>.

- Breiman, L. (2001). Random forests. Machine Learning 45:5-32, <doi:10.1023/A:1010933404324>.
- Malley, J. D., Kruppa, J., Dasgupta, A., Malley, K. G., & Ziegler, A. (2012). Probability machines: consistent probability estimation using nonparametric learning machines. Methods of Information in Medicine 51:74-81, <doi:10.3414/ME00010052>.
- Meinshausen (2006). Quantile Regression Forests. Journal of Machine Learning Research 7:983-999.

See Also

`predict.divfor`, `plot.interactionfor`, `plotEffects`

Examples

```
## Not run:

## Load package:

library("diversityForest")



## Set seed to make results reproducible:

set.seed(1234)



## Construct interaction forests and calculate EIM values:


# Binary outcome:
data(zoo)
modelcat <- interactionfor(dependent.variable.name = "type", data = zoo,
  num.trees = 20)


# Metric outcome:
data(stock)
modelcont <- interactionfor(dependent.variable.name = "company10", data = stock,
  num.trees = 20)


# Survival outcome:
library("survival")
mgus2$id <- NULL  # 'mgus2' data set is contained in the 'survival' package

# categorical variables need to be of factor format - important!!
mgus2$sex <- factor(mgus2$sex)
mgus2$pstat <- factor(mgus2$pstat)

# Remove the second time variable 'ptime':
mgus2$ptime <- NULL
```

```
# Remove missing values:
mgus2 <- mgus2[complete.cases(mgus2),]

# Take subset to make the calculations less computationally
# expensive for the example (in actual applications, we would of course
# use the whole data set):
mgus2sub <- mgus2[sample(1:nrow(mgus2), size=500),]

# Apply 'interactionfor':
modelsurv <- interactionfor(formula = Surv(futime, death) ~ ., data=mgus2sub, num.trees=20)

# NOTE: num.trees = 20 (in the above) would be much too small for practical
# purposes. This small number of trees was simply used to keep the
# runtime of the example short.
# The default number of trees is num.trees = 20000 if EIM values are calculated
# and num.trees = 2000 otherwise.



## Inspect the rankings of the variables and variable pairs with respect to
## the univariable, quantitative, and qualitative EIM values:

# Univariable EIM values:
modelcat$eim.univ.sorted

# Pairs with top quantitative EIM values:
modelcat$eim.quant.sorted[1:5]

# Pairs with top qualitative EIM values:
modelcat$eim.qual.sorted[1:5]



## Investigate visually the forms of the interaction effects of the variable pairs with
## largest quantitative and qualitative EIM values:

plot(modelcat)
plotEffects(modelcat, type="quant") # type="quant" is default.
plotEffects(modelcat, type="qual")



## Prediction:

# Separate 'zoo' data set randomly in training
# and test data:

data(zoo)
train.idx <- sample(nrow(zoo), 2/3 * nrow(zoo))
zoo.train <- zoo[train.idx, ]
zoo.test <- zoo[-train.idx, ]
```

```
# Construct interaction forest on training data:
# NOTE again: num.trees = 20 is specified too small for practical purposes.
modelcattrain <- interactionfor(dependent.variable.name = "type", data = zoo,
                                importance = "none", num.trees = 20)
# NOTE: Because we are only interested in prediction here, we do not
# calculate EIM values (by setting importance = "none"), because this
# speeds up calculations.

# Predict class values of the test data:
pred.zoo <- predict(modelcattrain, data = zoo.test)

# Compare predicted and true class values of the test data:
table(zoo.test$type, pred.zoo$predictions)

## End(Not run)
```

plot.interactionfor *Plot method for* interactionfor *objects*

Description

Plot function for interactionfor objects that allows to obtain a first overview of the result of the interaction forest analysis. This function visualises the distributions of the EIM values and the estimated forms of the bivariable influences of the variable pairs with largest quantitative and qualitative EIM values. Further visual exploration of the result of the interaction forest analysis should be conducted using plotEffects.

Usage

```
## S3 method for class 'interactionfor'
plot(x, numpairsquant = 2, numpairsqual = 2, ...)
```

Arguments

x	Object of class interactionfor.
numpairsquant	The number of pairs with largest quantitative EIM values to plot. Default is 2.
numpairsqual	The number of pairs with largest qualitative EIM values to plot. Default is 2.
...	Further arguments passed to or from other methods.

Details

For details on the plots of the estimated forms of the bivariable influences of the variable pairs see plotEffects.

NOTE: The p-values shown in the plots are generally much too optimistic and **MUST NOT** be reported as the result of a statistical test for significance of interaction. To obtain adjusted p-values that would correspond to valid tests, it would be possible to multiply these p-values by the number of possible variable pairs, which would correspond to Bonferroni-adjusted p-values. See the 'Details' section of `plotEffects` for further explanation and guidance. Note, however, that these Bonferroni-adjusted p-values should be interpreted with caution because, stemming from bivariable models, these p-values do not take the multivariable nature of the data into account.

NOTE ALSO: As described in Hornung & Boulesteix (2022), in the case of data with larger numbers of variables (larger than 100, but more seriously for high-dimensional data), the univariable EIM values can be biased. Therefore, it is strongly recommended to interpret the univariable EIM values with caution, if the data are high-dimensional. If it is of interest to measure the univariable importance of the variables for high-dimensional data, an additional conventional random forest (e.g., using the `ranger` package) should be constructed and the variable importance measure values of this random forest be used for ranking the univariable effects.

Value

A ggplot2 plot.

Author(s)

Roman Hornung

References

- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.

See Also

`plotEffects`

Examples

```
## Not run:

## Load package:

library("diversityForest")



## Set seed to make results reproducible:

set.seed(1234)



## Construct interaction forest and calculate EIM values:
```

```
data(stock)
model <- interactionfor(dependent.variable.name = "company10", data = stock,
                        num.trees = 20)

# NOTE: num.trees = 20 (in the above) would be much too small for practical
# purposes. This small number of trees was simply used to keep the
# runtime of the example short.
# The default number of trees is num.trees = 20000 if EIM values are calculated
# and num.trees = 2000 otherwise.



## When using the plot() function without further specifications,
## by default the estimated bivariable influences of the two pairs with largest quantitative
## and qualitative EIM values are shown:

plot(model)

# It is, however, also possible to change the numbers of
# pairs with largest quantitative and qualitative EIM values
# to be shown:

plot(model, numpairsquant = 4, numpairsqual = 3)


## End(Not run)
```

`plotEffects` *Interaction forest plots: exploring interaction forest results through visualisation*

Description

This function allows to visualise the (estimated) bivariable influences of pairs of variables (with large quantitative and qualitative EIM values) on the outcome. This step is crucial, because to interpret interaction effects between variable pairs with large quantitative and qualitative EIM values, it is necessary to learn about the forms these interaction effects take.

Usage

```
plotEffects(
  intobj,
  type = "quant",
  numpairs = 5,
  indpairs = NULL,
  pairs = NULL,
  allwith = NULL,
  pvalues = TRUE,
  twoplots = TRUE,
  addtitles = TRUE,
  plotit = TRUE
)
```

Arguments

`intobj`	Object of class `interactionfor`.
`type`	This can be either "quant" or "qual" and determines whether the plotted pairs are sorted according to either the quantitative or qualitative EIM values in decreasing order. Default is "quant".
`numpairs`	The number of pairs to plot (default: 5). This is overwritten by `indpairs`.
`indpairs`	Optional. The indices of the pairs in the sorted lists of quantitative (`type="quant"`) or qualitative EIM values to plot (`type="qual"`). This overwrites the `numpairs` argument.
`pairs`	This can be used to specify the pairs to plot. It is an optional list of character string vectors, where each of these vectors has length two. Each list element corresponds to one pair, where the first character string gives the name of the first member of the respective pair to plot and the second character string gives the name of the second member. This argument overwrites `numpairs` and `indpairs`.
`allwith`	This is an optional character string that can be set to the name of one of the variables. If provided, only variable pairs will be considered that feature the variable specified by this argument `allwith`. These pairs are again sorted in decreasing order according to the quantitative (`type="quant"`) or qualitative (`type="qual"`) EIM values and their number is restricted to the value given by `numpairs`. This argument `allwith` can be used, if it is of interest to learn whether a specific variable (e.g., sex or age) interacts with other variables in the data set and if so, which forms these interactions take.
`pvalues`	Set to `TRUE` (default) to add to the plots p-values from tests for interaction effect obtained using classical parametric regression approaches. For categorical outcomes logistic regression is used, for metric outcomes linear regression and for survival outcomes Cox regression. NOTE: These p-values are generally much too optimistic and **MUST NOT** be reported as the result of a statistical test for significance of interaction. See the 'Details' section below for further details.
`twoplots`	Set to `TRUE` / `FALSE` if for each plot page the results of two / one pair(s) of variables should be shown. Default is `TRUE`.
`addtitles`	Set to `TRUE` (default) to add headings providing the names of the variables in each pair. If `type="quant"`, these headings also give information on the type of quantitative interaction effect. Setting `addtitles` to `FALSE` is, for example, useful, when the produced plots are intended for use in a publication, where these headings might not be desirable.
`plotit`	This states whether the plots are actually plotted or merely returned as `ggplot` objects. Default is `TRUE`.

Details

For each considered pair the bivariable influence of both pair members on the outcome estimated using a two-dimensional flexible function is shown. Such visualisations make it possible to learn about the forms of the interaction effects between variable pairs with large EIM values. Moreover, these visualisations reveal (pathological) cases in which variable pairs do not show indications of interaction effects despite featuring large EIM values.
For binary outcomes the probabilities for the second class are estimated, for categorical outcomes with more than two classes the probabilities for the largest class (i.e., the class with the most observations) are estimated (using the function `plotPair`, a different class can be selected instead), for metric outcomes the means of the outcome are estimated, and for survival outcomes the log hazards ratio values compared to the median effect are estimated.
The kinds of estimates shown differ also according to whether both pair members are metric or only one of the two members is metric and the other one categorical or both pair members are categorical:

- If both pair members are metric and the outcome is categorical or metric we use two-dimensional LOESS regression, where in the case of categorical outcomes, to obtain probability estimates for the first class (or largest class for multi-class outcomes), we use the value '1' for the first class (largest class for multi-class outcomes) and the value '0' for the second class (all other classes for multi-class outcomes).
- If both pair members are metric and the outcome is survival we use a Cox proportional hazard additive model with a two-dimensional LOESS smooth (`gamcox` function from the 'MapGAM' package (version 1.2-5)) and in the rare cases for which the latter fails, we use classical Cox regression with an interaction term between the two covariates.
- If one pair member is metric and the other one categorical and the outcome is categorical or metric, we use LOESS regression between the outcome (coded as '0' and '1' in the case of categorical outcomes as described above) and the values of the metric variable separately for each category of the categorical variable. In the rare cases in which the LOESS regression fails we use classical linear regression.
- If one pair member is metric and the other one categorical and the outcome is survival, we use Cox regression with a linear tail-restricted cubic spline with four knots (univariable LOESS regression for survival outcomes does not seem to be available yet in R) separately for each category of the categorical variable. In cases in which the fitting of this spline regression fails we use classical Cox regression.
- If both pair members are categorical and the outcome is categorical or metric, we simply calculate the mean of the outcome (coded as '0' and '1' in the case of categorical outcomes as described above) for each possible combination of the categories of the two variables.
- If both pair members are categorical and the outcome is survival, we use classical Cox regression with an interaction term between the two variables (there is no need for any flexible modelling in this setting, because the Cox model with two categorical variables plus interaction term is saturated).

As described above (function argument: `pvalues`), there is an option to add p-values from tests for interaction effect to the plots. If at least one of the variables in the considered variable pair is categorical and features more than two categories, there are more than one interaction terms in the regression approaches used for testing, because the categorical variables are dummy-coded. Therefore, in these cases we obtain a p-value for each interaction term. to obtain a single p-value for the test for interaction we adjust these multiple p-values using the Holm-Bonferroni procedure and take the minimum of the adjusted p-values.

NOTE: These p-values are generally much too optimistic, in particular for small data sets and large numbers of variables. The reason for this overoptimism is that these p-values are not adjusted for the fact that we already used the data to find the variable pairs with strongest indications of interaction effects. This is similar to a multiple testing problem. Therefore, these p-values should only be seen as a rough guide to be interpreted very cautiously and **MUST NOT** be reported as the results of a statistical test for significance of interaction. To obtain adjusted p-values that would correspond to valid tests, it would be possible to multiply these p-values by the number of possible pairs, which would correspond to Bonferroni-adjusted p-values. For example, assume that we have 30 covariate variables. In that case the number of possible pairs would be 'choose(30, 2) = 435', which is why we would need to multiply each p-value by 435 to obtain an adjusted p-value (or keep the original p-values and divide the significance level 0.05 by 435). Note, however, that Bonferroni-adjusted p-values deliver quite conservative results, that is, weaker effects might not be detected using these p-values, while, however, effects for which these p-values are small (< 0.05) are most likely relevant. Note further that these Bonferroni-adjusted p-values should be interpreted with caution because, stemming from bivariable models, these p-values do not take the multivariable nature of the data into account.

Value

A list of ggplot2 plots returned invisibly.

Author(s)

Roman Hornung

References

- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.

See Also

`plot.interactionfor`, `plotPair`

Examples

```
## Not run:

## Load package:

library("diversityForest")



## Set seed to make results reproducible:
```

```
set.seed(1234)



## Construct interaction forest and calculate EIM values:

data(stock)
model <- interactionfor(dependent.variable.name = "company10", data = stock,
                        num.trees = 20)

# NOTE: num.trees = 20 (in the above) would be much too small for practical
# purposes. This small number of trees was simply used to keep the
# runtime of the example short.
# The default number of trees is num.trees = 20000 if EIM values are calculated
# and num.trees = 2000 otherwise.



## Obtain a first overview by applying the plot() function for
## interactionfor obects:

plot(model)



## Several possible application cases of the plotEffects() function:

# Visualise the estimated bivariable influences of the five variable pairs with the
# largest quantitative EIM values:

plotEffects(model) # type="quant" is default.


# Visualise the estimated bivariable influences of the five pairs with the
# largest qualitative EIM values:

plotEffects(model, type="qual")


# Visualise the estimated bivariable influences of all (eight) pairs that involve
# the variable "company7" sorted in decreasing order according to the
# qualitative EIM values:

plotEffects(model, allwith="company7", type="qual", numpairs=8)


# Visualise the estimated bivariable influences of the pairs with third and fifth
# largest qualitative EIM values:

plotEffects(model, type="qual", indpairs=c(3,5))


# Visualise the estimated bivariable influences of the pairs ("company3", "company5") and
# ("company1", "company9"):

plotEffects(model, pairs=list(c("company3", "company5"), c("company1", "company9")))



## Saving of plots generated with the plotEffects() function (e.g., for use in publications):

# Apply plotEffects() to obtain plots for the five variable pairs
# with the largest qualitative EIM values and store these plots in
# an object 'ps':

ps <- plotEffects(model, type="qual", pvalues=FALSE, twoplots=FALSE, addtitles=FALSE, plotit=FALSE)

# pvalues = FALSE states that no p-values should be shown in the plots,
# because these might not be desired in plots meant for publication.
# twoplots = FALSE ensures that we get one plot for each page instead of two plots per page.
# addtitles = FALSE removes the automatically generated titles, because these are likely
# not desired in publications.
# plotit = FALSE ensures that the plots are not displayed, but only returned (invisibly)
# by plotEffects().


# Save the plot with second largest qualitative EIM value:

p1 <- ps[[2]]

# Add title:
library("ggpubr")
p1 <- annotate_figure(p1, top = text_grob("My descriptive plot title 1", face = "bold", size = 14))
p1

# Save as PDF:
# library("ggplot2")
# ggsave(file="mypathtofolder/FigureXY1.pdf", width=14, height=6)


# Save the plot with fifth largest qualitative EIM value:

p2 <- ps[[5]]

# Add title:
p2 <- annotate_figure(p2, top = text_grob("My descriptive plot title 2", face = "bold", size = 14))
p2

# Save as PDF:
# ggsave(file="mypathtofolder/FigureXY1.pdf", width=14, height=6)


# Combine both of the above plots:
p <- ggarrange(p1, p2, nrow = 2)
p
```

```
# Save the combined plot:
# ggsave(file="mypathtofolder/FigureXYcombined.pdf", width=14, height=11)

# NOTE: Using plotEffects() it is not possible to change the plots
# themselves (by e.g., increasing the label sizes or changing the
# axes ranges). However, the function plotPair() can be used to change
# the plots themselves.


## End(Not run)
```

plotPair *Plot of the (estimated) simultaneous influence of two variables*

Description

This function allows to visualise the (estimated) bivariable influence of a single specific pair of variables on the outcome. The estimation and plotting is performed in the same way as in `plotEffects`. However, `plotPair` does not require an `interactionfor` object and can thus be used also without a constructed interaction forest.

Usage

```
plotPair(
  pair,
  yvarname,
  statusvarname = NULL,
  data,
  levelsorder1 = NULL,
  levelsorder2 = NULL,
  categprob = NULL,
  pvalue = TRUE,
  returnseparate = FALSE,
  intobj = NULL
)
```

Arguments

`pair`	Character string vector of length two, where the first character string gives the name of the first member of the respective pair to plot and the second character string gives the name of the second member. Note that the order of the two pair members in `pair` determines how the results are visualised: The estimated influence of the second pair member is visualised conditionally on different values of the first pair member.
`yvarname`	Name of outcome variable.
`statusvarname`	Name of status variable, only applicable to survival data.

`data`	Data frame containing the variables.
`levelsorder1`	Optional. Order the categories of the first variable should have in the plot (if it is categorical). Character string vector, where the i-th entry contains the name of the category that should take the i-th place in the ordering of the categories of the first variable.
`levelsorder2`	Optional. Order the categories of the second variable should have in the plot (if it is categorical). Character string vector specified in an analogous way as `levelsorder1`.
`categprob`	Optional. Only relevant for categorical outcomes with more than two classes. Name of the class for which probabilities should be estimated. As described in `plotEffects`, for categorical outcomes with more than two classes, by default the probabilities for the largest class (i.e., the class with the most observations) are estimated when visualising the bivariable influence of the variables. Using `categprob` a different class can be specified for the class for which probabilities should be estimated.
`pvalue`	Set to `TRUE` (default) to add to the plot a p-value from a test for interaction effect obtained using a classical parametric regression approach. For categorical outcomes logistic regression is used, for metric outcomes linear regression and for survival outcomes Cox regression. See the 'Details' section of `plotEffects` for further details.
`returnseparate`	Set to `TRUE` to return invisibly the two generated ggplot plots separately in the form of a list. The latter option is useful, because it allows to manipulate the resulting plots (label size etc.) and makes it possible to consider only one of the two plots. The default is `FALSE`, which results in the two plots being returned together in the form of a `ggarrange` object.
`intobj`	Optional. Object of class `interactionfor`. If this is provided, the ordering of the categories obtained when constructing the interaction forest will be used for categorical variables. See Hornung & Boulesteix (2022) for details.

Details

See the 'Details' section of `plotEffects`.

Value

A ggplot2 plot.

Author(s)

Roman Hornung

References

- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.

- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.

See Also

`plotEffects`, `plot.interactionfor`

Examples

```
## Not run:

## Load package:

library("diversityForest")



## Visualise the estimated bivariable influence of 'toothed' and 'feathers' on
## the probability of type="mammal":

data(zoo)
plotPair(pair = c("toothed", "feathers"), yvarname="type", data = zoo)



## Visualise the estimated bivariable influence of 'creat' and 'hgb' on
## survival (more precisely, on the log hazards ratio compared to the
## median effect):

library("survival")
mgus2compl <- mgus2[complete.cases(mgus2),]
plotPair(pair=c("creat", "hgb"), yvarname="futime", statusvarname = "death", data=mgus2compl)

# Problem: The outliers in the left plot make it difficult to see what is going
# on in the region with creat values smaller than about two even though the
# majority of values lie there.

# --> Solution: We re-run the above line setting returnseparate = TRUE, because
# this allows to get the two ggplot plots separately, which can then be manipulated
# to change the x-axis range in order to remove the outliers:

ps <- plotPair(pair=c("creat", "hgb"), yvarname="futime", statusvarname = "death",
               data=mgus2compl, returnseparate = TRUE)

# Change the x-axis range:
library("ggplot2")
ps[[1]] + xlim(c(0.5,2))
# Save the plot:
# ggsave(file="mypathtofolder/FigureXY1.pdf", width=7, height=6)

# We can, for example, also change the label sizes of the second plot:
```

```
# With original label sizes:
ps[[2]]
# With larger label sizes:
ps[[2]] +  theme(axis.title=element_text(size=15))
# Save the plot:
# library("ggplot2")
# ggsave(file="mypathtofolder/FigureXY2.pdf", width=7, height=6)


## End(Not run)
```

predict.divfor *Diversity Forest prediction*

Description

Prediction with new data and a saved forest from divfor.

Usage

```
## S3 method for class 'divfor'
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),
  seed = NULL,
  num.threads = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	`divfor` 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.
`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. Default is number of CPUs available.
`verbose`	Verbose output on or off.
`...`	further arguments passed to or from other methods.

Details

This package is a fork of the R package 'ranger' that implements random forests using an efficient C++ implementation. More precisely, 'diversityForest' was written by modifying the code of 'ranger', version 0.11.0. Therefore, details on further functionalities of the code that are not presented in the help pages of 'diversityForest' are found in the help pages of 'ranger' (version 0.11.0). The code in the example sections of `divfor` and `tunedivfor` can be used as a template for all common application scenarios with respect to classification, regression and survival prediction using univariable, binary splitting. Some function arguments adopted from the 'ranger' package may not be useable with diversity forests (for the current package version).

Value

Object of class `divfor.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

- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 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. Journal of Machine Learning Research 15:1625-1651.

- Meinshausen (2006). Quantile Regression Forests. Journal of Machine Learning Research 7:983-999.

See Also

`divfor`

`predict.interactionfor` *Interaction Forest prediction*

Description

Prediction with new data and a saved interaction forest from `interactionfor`.

Usage

```
## S3 method for class 'interactionfor'
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),
  seed = NULL,
  num.threads = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	`interactionfor` 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.
`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. Default is number of CPUs available.
`verbose`	Verbose output on or off.
`...`	further arguments passed to or from other methods.

Details

Note that his package is a fork of the R package 'ranger' that implements random forests using an efficient C++ implementation. The documentation is in large parts taken from 'ranger', where some parts of the documentation may not apply to (the current version of) the 'diversityForest' package. Details on further functionalities of the code that are not presented in the help pages of 'diversityForest' are found in the help pages of 'ranger' (version 0.11.0).

Value

Object of class `interaction.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, Roman Hornung

References

- Hornung, R., Boulesteix, A.-L. (2022). Interaction forests: Identifying and exploiting interpretable quantitative and qualitative interaction effects. Computational Statistics & Data Analysis 171:107460, <doi:10.1016/j.csda.2022.107460>.
- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 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. Journal of Machine Learning Research 15:1625-1651.
- Meinshausen (2006). Quantile Regression Forests. Journal of Machine Learning Research 7:983-999.

See Also

`interactionfor`

`stock` *Data on stock prices of aerospace companies*

Description

This data set contains 950 daily stock prices from January 1988 through October 1991, for ten aerospace companies. The names of the companies are anonymised and the stock prices for one of these companies (`company10`) were flagged as the outcome variable. Thus, for this data set, both the outcome and the covariates were metric.

Format

A data frame with 950 observations, nine covariates and one metric outcome variable

Details

The variables are as follows: covariates: `company1`, ..., `company9`, outcome variable: `company10`.

Source

OpenML: data.name: stock, data.id: 223, link: `https://www.openml.org/d/223/`

References

- Vanschoren, J., van Rijn, J. N., Bischl, B., Torgo, L. (2013). OpenML: networked science in machine learning. SIGKDD Explorations 15(2):49-60, <doi:10.1145/2641190.2641198>.

Examples

```
## Load data:
data(stock)

## Dimension of data:
dim(stock)

## First rows of data:
head(stock)
```

`tunedivfor`	*Optimization of the values of the tuning parameters* `nsplits` *and* `proptry`

Description

First, both for `nsplits` and `proptry` a grid of possible values may be provided, where default grids are used if no grids are provided. Second, for each pairwise combination of values from these two grids a forest is constructed. Third, that pair of `nsplits` and `proptry` values is used as the optimized set of parameter values that is associated with the smallest out-of-bag prediction error. If several pairs of parameter values are associated with the same smallest out-of-bag prediction error, the pair with the smallest (parameter) values is used.

Usage

```
tunedivfor(
  formula = NULL,
  data = NULL,
  nsplitsgrid = c(2, 5, 10, 30, 50, 100, 200),
  proptrygrid = c(0.05, 1),
  num.trees.pre = 500
)
```

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).
`nsplitsgrid`	Grid of values to consider for `nsplits`. Default grid: 2, 5, 10, 30, 50, 100, 200.
`proptrygrid`	Grid of values to consider for `proptry`. Default grid: 0.05, 1.
`num.trees.pre`	Number of trees used for each forest constructed during tuning parameter optimization. Default is 500.

Value

List with elements

`nsplitsopt`	Optimized value of `nsplits`.
`proptryopt`	Optimized value of `proptry`.
`tunegrid`	Two-dimensional `data.frame`, where each row contains one pair of values considered for `nsplits` (first entry) and `proptry` (second entry).
`ooberrs`	The out-of-bag prediction errors obtained for each pair of values considered for `nsplits` and `proptry`, where the ordering of pairs of values is the same as in `tunegrid` (see above).

Author(s)

Roman Hornung

References

- Hornung, R. (2022). Diversity forests: Using split sampling to enable innovative complex split procedures in random forests. SN Computer Science 3(2):1, <doi:10.1007/s42979021-009201>.
- Wright, M. N., Ziegler, A. (2017). ranger: A fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 77:1-17, <doi:10.18637/jss.v077.i01>.

See Also

`divfor`

Examples

```
## Load package:

library("diversityForest")


## Set seed to obtain reproducible results:

set.seed(1234)


## Tuning parameter optimization for the iris data set:

tuneres <- tunedivfor(formula = Species ~ ., data = iris, num.trees.pre = 20)
# NOTE: num.trees.pre = 20 is specified too small for practical
# purposes - the out-of-bag error estimates of the forests
# constructed during optimization will be much too variable!!
# In practice, num.trees.pre = 500 (default value) or a
# larger number should be used.

tuneres

tuneres$nsplitsopt
tuneres$proptryopt
tuneres$tunegrid
tuneres$ooberrs
```

zoo *Data on biological species*

Description

This data set describes 101 different biological species using 16 simple attributes, where 15 of these are binary and one is metric (the number of legs). The outcome "mammal vs. other" (`type`) is binary.

Format

A data frame with 101 observations, 16 covariates and one binary outcome variable

Details

The variables are as follows:

- `hair`. factor. Presence of hairs (true = yes; false = no)
- `feathers`. factor. Presence of feathers (true = yes; false = no)
- `eggs`. factor. Does the species lay eggs? (true = yes; false = no)
- `milk`. factor. Does the species give milk? (true = yes; false = no)
- `airborne`. factor. Does the species fly? (true = yes; false = no)
- `aquatic`. factor. Does the species live in the water? (true = yes; false = no)
- `predator`. factor. Is the species a predator? (true = yes; false = no)
- `toothed`. factor. Presence of teeth (true = yes; false = no)
- `backbone`. factor. Presence of backbone (true = yes; false = no)
- `breathes`. factor. Does the species breathe with lungs? (true = yes; false = no)
- `venomous`. factor. Is the species venomous? (true = yes; false = no)
- `fins`. factor. Presence of fins (true = yes; false = no)
- `legs`. metric. Number of legs
- `tail`. factor. Presence of tail (true = yes; false = no)
- `domestic`. factor. Is the species domestic? (true = yes; false = no)
- `catsize`. factor. Is the species large? (true = yes; false = no)
- `type`. factor. Binary outcome variable - type of species ('mammal' vs. 'other')

The original openML dataset contains an additional variable `animal`, which is removed in this version of the data set. This variable provided the names of all species.

Source

OpenML: data.name: zoo, data.id: 965, link: https://www.openml.org/d/965/

References

- Vanschoren, J., van Rijn, J. N., Bischl, B., Torgo, L. (2013). OpenML: networked science in machine learning. SIGKDD Explorations 15(2):49-60, <doi:10.1145/2641190.2641198>.
- Dua, D., Graff, C. (2019) UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. `https://archive.ics.uci.edu/ml/`.

Examples

```
##' Load data:
data(zoo)

##' Numbers of observations in the two classes:
table(zoo$type)

##' Dimension of data:
dim(zoo)

##' First rows of data:
head(zoo)
```

Index