# Package 'diversityForest'

September 16, 2024

**Type** Package

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

**Version** 0.5.0

**Date** 2024-09-16

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

**Description** Implementations of three diversity forest (DF) (Hornung, 2022, <doi:10.1007/s42979-021-00920-1>) variants.
The DF algorithm is a split-finding approach that allows complex split procedures to be realized in random forest variants.
The three DF variants implemented are:

1. interaction forests (IFs) (Hornung & Boulesteix, 2022, <doi:10.1016/j.csda.2022.107460>): Model quantitative and qualitative interaction effects using bivariable splitting. Come with the Effect Importance Measure (EIM), which can be used to identify variable pairs that have well-interpretable quantitative and qualitative interaction effects with high predictive relevance.
2. multi forests (MuFs) (Hornung & Hapfelmeier, 2024, <doi:10.48550/arXiv.2409.08925>): Model multi-class outcomes using multi-way and binary splitting. Come with two variable importance measures (VIMs): The multi-class VIM measures the degree to which the variables are specifically associated with one or more outcome classes, and the discriminatory VIM, similar to conventional VIMs, measures the overall influence strength of the variables.
3. the basic form of diversity forests that uses conventional univariable, binary splitting (Hornung, 2022).

Except for multi forests, which are tailored for multi-class outcomes, all included diversity forest variants support categorical, metric, and survival outcomes.
The package also includes plotting functions that make it possible to learn about the forms of the effects identified using IFs and MuFs.
This is a fork of the R package 'ranger' (main author: Marvin N. Wright), which implements random forests using an efficient C++ implementation.

**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, patchwork

**LinkingTo** Rcpp, RcppEigen

**Depends** R (>= 3.5)

**Suggests** testthat, BOLTSSIRR

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

**RoxygenNote** 7.3.1

**NeedsCompilation** yes

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

**Repository** CRAN

**Date/Publication** 2024-09-16 15:00:08 UTC

# Contents

`diversityForest-package` *Diversity Forests*

## Description

The diversity forest algorithm is a split-finding approach that allows complex split procedures to be realized in random forest variants. This is achieved by drastically reducing the numbers of candidate splits that need to be evaluated for each split. The algorithm 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 three 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).
- *multi forests (MuFs)* (Hornung & Hapfelmeier, 2024), a diversity forest-variant for multi-class outcomes. MuFs use both multi-way and binary splitting. The latter form the basis for the *multi-class variable importance measure (VIM)* and the *discriminatory VIM* associated with MuFs. The multi-class VIM measures the degree to which the variables are specifically associated with one or several of the outcome classes. In contrast, the discriminatory VIM, similar to conventional VIMs, measures the general influence of the variables regardless of their specific association with individual classes.

Diversity forests with univariable, binary splitting can be constructed using the function `divfor`, interaction forests using the function `interactionfor`, and multi forests using the function `multifor`. Except for multi forests, which are tailored for multi-class outcomes, all included diversity forest variants 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>.
- Hornung, R., Hapfelmeier, A. (2024). Multi forests: Variable importance for multi-class outcomes. arXiv:2409.08925, <doi:10.48550/arXiv.2409.08925>.
- 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>.


---

`ctg` *Data on automatic analysis of cardiotocograms*

---

## Description

This data set contains measurements from 2126 fetal cardiotocograms (CTGs). The CTGs were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. This description is taken from the UC Irvine Machine Learning Repository, where this data set was downloaded from. The outcome `CLASS` is categorical with ten classes that correspond to different fetal heart rate patterns. See the 'Details' section below for further information.

## Format

A data frame with 2126 observations, 25 covariates and one 10-class outcome variable

## Details

The variables are as follows:

- `b`. numeric. Start instant
- `e`. numeric. End instant
- `LBE`. numeric. Fetal heart rate (FHR) baseline value assessed by medical expert (beats per minute)
- `LB`. numeric. FHR baseline value assessed by SisPorto (beats per minute)
- `AC`. numeric. Number of accelerations per second
- `FM`. numeric. Number of fetal movements per second
- `UC`. numeric. Number of uterine contractions per second

- DL. numeric. Number of light decelerations per second
- DS. numeric. Number of severe decelerations per second
- DP. numeric. Number of prolonged decelerations per second
- DR. numeric. Number of repetitive decelerations per second
- ASTV. numeric. Percentage of time with abnormal short term variability
- MSTV. numeric. Mean value of short term variability
- ALTV. numeric. Percentage of time with abnormal long term variability
- MLTV. numeric. Mean value of long term variability
- Width. numeric. Width of FHR histogram
- Min. numeric. Minimum value of FHR histogram
- Max. numeric. Maximum value of FHR histogram
- Nmax. numeric. Number of histogram peaks
- Nzeros. numeric. Number of histogram zeros
- Mode. numeric. Mode of the histogram
- Mean. numeric. Mean of the histogram
- Median. numeric. Median of the histogram
- Variance. numeric. Variance of the histogram
- Tendency. factor. Histogram tendency (-1 for left asymmetric; 0 for symmetric; 1 for right asymmetric)
- CLASS. factor. FHR pattern class

The classes of the outcome CLASS are as follows:

- A. Calm sleep
- B. REM sleep
- C. Calm vigilance
- D. Active vigilance
- SH. Shift pattern (A or Susp with shifts)
- AD. Accelerative/decelerative pattern (stress situation)
- DE. Decelerative pattern (vagal stimulation)
- LD. Largely decelerative pattern
- FS. Flat-sinusoidal pattern (pathological state)
- SUSP. SUSP suspect pattern

This is a pre-processed version of the "Cardiotocography" data set published in the UC Irvine Machine Learning Repository. The raw data contained four additional variables Date, FileName, SegFile, and NSP, which were removed in this version of the data.

## Source

UC Irvine Machine Learning Repository, link: https://archive.ics.uci.edu/dataset/193/cardiotocography/ (Accessed: 29/08/2024)

## References


- Ayres-de Campos, D., Bernardes, J., Garrido, A., Marques-de-Sá, J., Pereira-Leite, L. (2000). SisPorto 2.0: a program for automated analysis of cardiotocograms. J Matern Fetal Med. 9(5):311-318, <doi:10.1002/15206661(200009/10)9:5<311::AIDMFM12>3.0.CO;29>.
- 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(ctg)

# Numbers of observations per outcome class:
table(ctg$CLASS)

# Dimension of data:
dim(ctg)

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

---

`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 misclassified 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)
```

hars *Data on human activity recognition using smartphones*

## Description

This data set contains sensor data from 30 volunteers aged 19-48 years, performing six activities while wearing Samsung Galaxy S II smartphones on their waists. The sensors recorded 3-axial linear acceleration and angular velocity at 50Hz. The experiments were video-recorded to label the data manually. The outcome `Activity` is categorical with six classes that differentiate the six activities.
This is an updated version of the Human Activity Recognition Using Smartphones data set published in the UC Irvine Machine Learning Repository. This updated version published on OpenML includes both raw sensor signals and updated activity labels, with aggregated measurements for each individual and activity.

## Format

A data frame with 180 observations (activities), 66 covariates and one 6-class outcome variable

## Details

The classes of the outcome `Activity` are as follows: `LAYING`, `SITTING`, `STANDING`, `WALKING`, `WALKING_DOWNSTAIRS`, `WALKING_UPSTAIRS`.
The OpenML data set contained one additional variable `Person` that was removed because it has too many factors to use it as a covariate in prediction.

## Source

- Updated version: OpenML: data.name: Smartphone-Based_Recognition_of_Human_Activities, data.id: 4153, link: https://www.openml.org/d/4153/ (Accessed: 29/08/2024)
- Original version: UC Irvine Machine Learning Repository, link: https://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones/ (Accessed: 29/08/2024)

## References


- Reyes-Ortiz, J.-L., Oneto, L., Samà, A., Parra, X., Anguita, D. (2016). Transition-aware human activity recognition using smartphones. Neurocomputing, 171:754-767, <doi:10.1016/j.neucom.2015.07.085>.
- 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(hars)

# Numbers of observations per outcome class:
table(hars$Activity)

# Dimension of data:
dim(hars)

# First rows of (subset) data:
head(hars[,1:5])
```

---

`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.train,
                                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)
```

---

`multifor` *Construct a multi forest prediction rule and calculate multi-class and discriminatory variable importance scores as described in Hornung & Hapfelmeier (2024).*

---

## Description

Implements multi forests, a random forest variant tailored for multi-class outcomes (Hornung & Hapfelmeier, 2024). Multi forests feature the multi-class variable importance measure (VIM) and the discriminatory VIM.
The *multi-class VIM* measures the degree to which the variables are specifically associated with one or more classes. In contrast, conventional VIMs, such as the permutation VIM or the Gini importance, measure the overall influence of variables regardless of their class-association. Therefore, these measures rank not only class-associated variables high, but also variables that only discriminate well between groups of classes. This is problematic, if only class-associated variables are to be identified.
Similar to conventional VIMs, the *discriminatory VIM* measures the general influence of the variables.

NOTE: To learn about the shapes of the influences of the variables with the largest multi-class VIM values on the multi-class outcome, it is crucial to apply the `plot.multifor` function to the `multifor` object. Two further plot functions are `plotMcl` and `plotVar`.

NOTE also: The purpose of the multi forest algorithm is mainly to calculate the multi-class VIM values. A large-scale real data comparison study in Hornung & Hapfelmeier (2024) revealed that multi forests often have a slightly lower predictive performance than conventional random forests. This was especially true with respect to calibration and for data sets with many outcome classes. Therefore, if it is important to maximize the predictive performance or for data sets with many classes, for prediction other classifiers than multi forests (e.g. conventional random forests) should be explored.

## Usage

```
multifor(
  formula = NULL,
  data = NULL,
  num.trees = ifelse(nrow(data) <= 5000, 5000, 1000),
  importance = "both",
  write.forest = TRUE,
  probability = TRUE,
  min.node.size = NULL,
  max.depth = NULL,
  replace = FALSE,
  sample.fraction = ifelse(replace, 1, 0.7),
  case.weights = NULL,
  keep.inbag = FALSE,
  inbag = NULL,
  holdout = FALSE,
  oob.error = TRUE,
  num.threads = NULL,
  verbose = TRUE,
  seed = NULL,
  dependent.variable.name = NULL,
  mtry = NULL,
  npervar = 5
)
```

## 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, or matrix, dgCMatrix (Matrix). |
| num.trees | Number of trees. Default is 5000 for datasets with a maximum of 5000 observations and 1000 for datasets with more than 5000 observations. |
| importance | Variable importance mode, one of the following: "both" (the default), "multiclass", "discriminatory", "none". If "multiclass", multi-class VIM values are computed, if "discriminatory", discriminatory VIM values are computed, and if "both", both multi-class and discriminatory VIM values are computed. See the 'Details' section below for details. |
| write.forest | Save multifor.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). Using this option (default is TRUE), class probability predictions are obtained. |
| min.node.size | Minimal node size. Default 5 for probability and 1 for classification. |
| 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. 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.

`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.

`oob.error`
: Compute OOB prediction error. Default is `TRUE`.

`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.

`mtry`
: Number of candidate variables to sample for each split. Default is the (rounded down) square root of the number variables.

`npervar`
: Number of splits to sample per candidate variable. Default is 5.

## Details

The multi-class VIM is only calculated for variables that feature at least as many unique values as there are outcome classes.
Before learning the multi forest, the categories of unordered categorical variables are ordered using an approach by Coppersmith et al. (1999), which ensures that close categories feature similar outcome class distributions. This approach is also used in the `ranger` R package, when using the option `respect.unordered.factors="order"`.

## Value

Object of class `multifor` with elements

`predictions`
: Predicted classes (for `probability=FALSE`) or class probabilities (for `probability=TRUE`), based on out-of-bag samples.

`num.trees`
: Number of trees.

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

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

`mtry`
: Number of candidate variables sampled for each split.

`var.imp.multiclass`
: Multi-class VIM values. Only computed for independent variables that feature at least as many unique values as the outcome variable has classes. For other variables, the entries in the vector `var.imp.multiclass` will be `NA`.

`var.imp.discr`
: Discriminatory VIM values for all independent variables.

`prediction.error`
: Overall out-of-bag prediction error. For classification this is the fraction of misclassified samples and for probability estimation the Brier score.

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

`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.

`treetype`
: Type of forest/tree. Classification or probability.

`call`
: Function call.

`importance.mode`
: Importance mode used.

`num.samples`
: Number of samples.

`replace`
: Sample with replacement.

`plotres`
: List ob objects needed by the plot functions: `data` contains the data; `yvarname` is the name of the outcome variable.

## Author(s)

Roman Hornung, Marvin N. Wright

## References


- Hornung, R., Hapfelmeier, A. (2024). Multi forests: Variable importance for multi-class outcomes. arXiv:2409.08925, <doi:10.48550/arXiv.2409.08925>.
- 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>.
- Coppersmith, D., Hong, S. J., Hosking, J. R. (1999). Partitioning nominal attributes in decision trees. Data Mining and Knowledge Discovery 3:197-217, <doi:10.1023/A:1009869804967>.


## See Also

`predict.multifor`

## Examples

```
## Not run:

## Load package:

library("diversityForest")



## Set seed to make results reproducible:

set.seed(1234)



## Load the "ctg" data set:

data(ctg)



## Construct a multi forest:

model <- multifor(dependent.variable.name = "CLASS", data = ctg,
                  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 = 5000 for datasets with a maximum of
# 5000 observations and num.trees = 1000 for datasets larger than that.



## The out-of-bag estimated Brier score (note that by default
## 'probability = TRUE' is used in 'multifor'):

model$prediction.error



## Inspect the multi-class and the discriminatory VIM values:

model$var.imp.multiclass

# --> Note that there are no multi-class VIM values for some of the variables.
# These are those for which there are fewer unique values than outcome classes.
# See the "Details" section above.

model$var.imp.discr
```

```
## Inspect the 5 variables with the largest multi-class VIM values and the
## 5 variables with the largest discriminatory VIM values:

sort(model$var.imp.multiclass, decreasing = TRUE)[1:5]

sort(model$var.imp.discr, decreasing = TRUE)[1:5]



## Instead of passing the name of the outcome variable through the
## 'dependent.variable.name' argument as above, the formula interface can also
## be used. Below, we fit a multi forest with only the first five variables
## from the 'ctg' data set:

model <- multifor(CLASS ~ b + e + LBE + LB + AC, data=ctg, num.trees = 20)


## As expected, the out-of-bag estimated prediction error is much larger
## for this model:

model$prediction.error



## NOTE: Visual exploration of the results of the multi-class VIM analysis
## is crucial.
## Therefore, in practice the next step would be to apply the
## 'plot.multifor' function to the object 'model'.

# plot(model)




## Prediction:


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

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

# Construct multi forest on training data:
# NOTE again: num.trees = 20 is specified too small for practical purposes.
model_train <- multifor(dependent.variable.name = "CLASS", data = ctg.train,
                        importance = "none", probability = FALSE,
                        num.trees = 20)
# NOTE: Because we are only interested in prediction here, we do not
# calculate VIM values (by setting importance = "none"), because this
```

```
# speeds up calculations.
# NOTE also: Because we are interested in class label prediction here
# rather than class probability prediction we specified 'probability = FALSE'
# above.

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

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



## Repeat the analysis for class probability prediction
## (default 'probability = TRUE'):

model_train <- multifor(dependent.variable.name = "CLASS", data = ctg.train,
                        importance = "none", num.trees = 20)

# Predict class probabilities in the test data:
pred.ctg <- predict(model_train, data = ctg.test)

# The predictions are now a matrix of class probabilities:
head(pred.ctg$predictions)

# Obtain class predictions by choosing the classes with the maximum predicted
# probabilities (the function 'which.is.max' chooses one class randomly if
# there are several classes with maximum probability):
library("nnet")
classes <- levels(ctg.train$CLASS)
pred_classes <- factor(classes[apply(pred.ctg$predictions, 1, which.is.max)],
                       levels=classes)

# Compare predicted and true class values of the test data:
table(ctg.test$CLASS, pred_classes)


## 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)
```

plot.multifor *Plot method for* multifor *objects*

## Description

Plot function for multifor objects that allows to obtain a first overview of the result of the multi-class VIM analysis. This function visualises the distribution of the multi-class VIM values together with that of the corresponding discriminatory VIM values and the estimated dependency structures of the multi-class outcome on the variables with largest multi-class VIM values. These estimated dependency structures are visualised using density plots and/or boxplots.

## Usage

```
## S3 method for class 'multifor'
plot(x, plot_type = c("both", "density", "boxplot")[1], num_best = 5, ...)
```

## Arguments

| | |
|---|---|
| x | Object of class multifor. |
| plot_type | Plot type, one of the following: "both" (the default), "density", "boxplot". If "density", "density" plots are produced, if "boxplot", "boxplot" plots are produced, and if "both", both "density" plots and "boxplot" plots are produced. See the 'Details' section of plotMcl for details. |
| num_best | The number of variables with largest multi-class VIM values to plot. Default is 5. |
| ... | Further arguments passed to or from other methods. |

## Details

In the plot showing the distribution of the multi-class VIM values along with that of the discriminatory VIM values, the discriminatory VIM values are normalized to make them comparable to the multi-class VIM values. This is achieved by dividing the discriminatory VIM values by their mean and multiplying it by that of the multi-class VIM values. Although the discriminatory VIM values are computed for all variables, only those variables for which the multi-class VIM values were computed are included in this analysis (i.e., all variables that have at least as many unique values as there are classes in the outcome variable).
For details on the plots of the estimated dependency structures of the multi-class outcome on the variables, see plotMcl. The latter function allows to visualise these estimated dependency structures for arbitrary variables in the data.

## Value

A ggplot2 plot.

## Author(s)

Roman Hornung

## References


- Hornung, R., Hapfelmeier, A. (2024). Multi forests: Variable importance for multi-class outcomes. arXiv:2409.08925, <doi:10.48550/arXiv.2409.08925>.
- 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

plotMcl

## Examples

```
## Not run:

## Load package:

library("diversityForest")



## Set seed to make results reproducible:

set.seed(1234)



## Construct multi forest and calculate multi-class and discriminatory VIM values:

data(hars)
model <- multifor(dependent.variable.name = "Activity", data = hars,
                  num.trees = 100, probability=TRUE)

# NOTE: num.trees = 100 (in the above) would be likely 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 = 5000 for datasets with a maximum of
# 5000 observations and num.trees = 1000 for datasets larger than that.



## By default the estimated class-specific distributions of the num_best=5
## variables with the largest multi-class VIM values are plotted:

plot(model)

## Consider only the 2 variables with the largest multi-class VIM values:

plot(model, num_best = 2)

## Show only the density plots or only the boxplots:
```

```
plot(model, plot_type = "density", num_best = 2)
plot(model, plot_type = "boxplot", num_best = 2)

## Show only the plot of the distributions of the multi-class and
## discriminatory VIM values:

plot(model, num_best = 0)


## 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)
```

plotMcl *Plots of the (estimated) within-class distributions of variables*

## Description

This function allows to visualise the (estimated) distributions of one or several variables for each of the classes of the outcomes. This allows to study how exactly variables of interest influence the outcome, which is crucial for interpretive purposes. Two types of visualisations are available: density plots and boxplots. See the 'Details' section below for further explanation.

## Usage

```
plotMcl(
  data,
  yvarname,
  varnames,
  plot_type = c("both", "density", "boxplot")[1],
  addtitles = TRUE,
  plotit = TRUE
)
```

## Arguments

| | |
|---|---|
| data | Data frame containing the variables. |
| yvarname | Name of outcome variable. |
| varnames | Names of the variables for which plots should be created. |
| plot_type | Plot type, one of the following: "both" (the default), "density", "boxplot". If "density", "density" plot are produced, if "boxplot", "boxplot" plots are produced, and if "both", both "density" plots and "boxplot" plots are produced. See the 'Details' section below for details. |
| addtitles | Set to TRUE (default) to add headings providing the names of the respective variables to the plots. |
| plotit | This states whether the plots are actually plotted or merely returned as ggplot objects. Default is TRUE. |

## Details

For the "density" plots, kernel density estimates (obtained using the density() function from base R) of the within-class distributions are plotted in the same plot using different colors and, depending on the number of classes, different line types. To account for the different number of observations per class, each density is multiplied by the proportion of observations from that class. The resulting scaled densities can be interpreted in terms of the local density of the observations from each class relative to those from the other classes. For example, if a scaled density has the largest value in a particular region, this can be interpreted as the respective class being the most frequent in that region. Another example: If the scaled density of class "A" is twice as large as the

scaled density of class "B" in a particular region, this can be interpreted to mean that there are twice as many observations of class "A" as of class "B" in that region.

In the `"density"` plots, only classes represented by at least two observations are considered. If the number of classes is greater than 7, the different classes are distinguished using both colors and line styles. To indicate the absolute numbers of observations in the different regions, the locations of the observations from the different classes are visualized using a rug plot on the x-axis, using the same colors and line types as for the density plots. If the number of observations is greater than 1,000, a random subset of 1,000 observations is shown in the rug plot instead of all observations for visual clarity.

The `"boxplot"` plots show the (estimated) within-class distributions side by side using boxplots. All classes are considered, even those represented by only a single observation. For the `plot_type="both"` option, which displays both `"density"` and `"boxplot"` plots, the boxplots are displayed using the same colors and ( if applicable) line styles as the kernel density estimates, for clarity. Boxplots of classes for which no kernel density estimates were obtained (i.e., those of the classes represented by single observations) are shown in grey.

Note that plots are only generated for those variables in `varnames` that have at least as many unique values as there are outcome classes. For categorical variables, the category labels are printed on the x- or y-axis of the `"density"` or `"boxplot"` plots, respectively. The rug plots of the `"density"` plots are produced only for numeric variables.

## Value

A list of ggplot2 plots returned invisibly.

## Author(s)

Roman Hornung

## References


- Hornung, R., Hapfelmeier, A. (2024). Multi forests: Variable importance for multi-class outcomes. arXiv:2409.08925, <doi:10.48550/arXiv.2409.08925>.
- 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.multifor`, `plotVar`

## Examples

```
## Not run:

## Load package:

library("diversityForest")
```

```
## Plot "density" and "boxplot" plots (default: plot_type = "both") for the
## first three variables in the "hars" dataset:

data(hars)
plotMcl(data = hars, yvarname = "Activity", varnames = c("tBodyAcc.mean...X",
                                                         "tBodyAcc.mean...Y",
                                                         "tBodyAcc.mean...Z"))


## Plot only the "density" plots for these variables:

plotMcl(data = hars, yvarname = "Activity",
        varnames = c("tBodyAcc.mean...X", "tBodyAcc.mean...Y",
                     "tBodyAcc.mean...Z"), plot_type = "density")

## Plot the "density" plots for these variables, but without titles of the
## plots:

plotMcl(data = hars, yvarname = "Activity", varnames =
          c("tBodyAcc.mean...X", "tBodyAcc.mean...Y", "tBodyAcc.mean...Z"),
        plot_type = "density", addtitles = FALSE)


## Make density plots for these variables, but only save them in a list "ps"
## without plotting them ("plotit = FALSE"):

ps <- plotMcl(data = hars, yvarname = "Activity", varnames =
                c("tBodyAcc.mean...X", "tBodyAcc.mean...Y",
                  "tBodyAcc.mean...Z"), plot_type = "density",
              addtitles = FALSE, plotit = FALSE)


## The plots can be manipulated later by using ggplot2 functionalities:

library("ggplot2")
p1 <- ps[[1]] + ggtitle("First variable in the dataset") +
  labs(x="Variable values", y="my scaled density")

p2 <- ps[[3]] + ggtitle("Third variable in the dataset") +
  labs(x="Variable values", y="my scaled density")


## Combine both of the above plots:

library("ggpubr")
p <- ggarrange(p1, p2, ncol = 2)
p

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

```
## 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)
```

---

`plotVar` *Plot of the (estimated) dependency structure of a variable* `x` *on a categorical variable* `y`

---

## Description

This function allows to visualise the (estimated) distributions of a variable `x` for each of the categories of a categorical variable `y`. This allows to study the dependency structure of `y` on `x`. Two types of visualisations are available: density plots and boxplots.

## Usage

```
plotVar(
  x,
  y,
  plot_type = c("both", "density", "boxplot")[1],
  x_label = "",
  y_label = "",
  plot_title = ""
)
```

## Arguments

| | |
|---|---|
| `x` | Metric variable or ordered categorical variable that has at least as many unique values as `y` |
| `y` | Factor variable with at least three categories. |
| `plot_type` | Plot type, one of the following: "both" (the default), "density", "boxplot". If "density", a `"density"` plot is produced, if "boxplot", a `"boxplot"` is produced, and if "both", both a `"density"` plot and a `"boxplot"` are produced. See the 'Details' section of `plotMcl` for details. |
| `x_label` | Optional. The label of the x-axis. |
| `y_label` | Optional. The label (heading) of the legend that differentiates the categories of `y`. |
| `plot_title` | Optional. The title of the plot. |

## Details

See the 'Details' section of `plotMcl`.

## Value

A ggplot2 plot.

## Author(s)

Roman Hornung

## References


- Hornung, R., Hapfelmeier, A. (2024). Multi forests: Variable importance for multi-class outcomes. arXiv:2409.08925, <doi:10.48550/arXiv.2409.08925>.
- 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

plotMcl, plot.multifor

## Examples

```
## Not run:

## Load package:

library("diversityForest")



## Load the "ctg" data set:

data(ctg)


## Set seed to make results reproducible (this is necessary because
## the rug plot produced by 'plotVar' does not show all observations, but
## only a random subset of 1000 observations):

set.seed(1234)


## Using a "density" plot and a "boxplot", visualise the (estimated)
## distributions of  the variable "Mean" for each of the categories of the
# variable "Tendency":

plotVar(x = ctg$Mean, y = ctg$Tendency)


## Re-create this plot with labels:

plotVar(x = ctg$Mean, y = ctg$Tendency, x_label = "Mean of the histogram ('Mean')",
        y_label = "Histogram tendency ('Tendency')",
        plot_title = "Relationship between 'Mean' and 'Tendency'")
```

```
## Re-create this plot, but only show the "density" plot:

plotVar(x = ctg$Mean, y = ctg$Tendency, plot_type = "density",
        x_label = "Mean of the histogram ('Mean')",
        y_label = "Histogram tendency ('Tendency')",
        plot_title = "Relationship between 'Mean' and 'Tendency'")


## Use ggplot2 and RColorBrewer functionalities to change the line colors and
## the labels of the categories of "Tendency":

library("ggplot2")
library("RColorBrewer")
p <- plotVar(x = ctg$Mean, y = ctg$Tendency, plot_type = "density",
             x_label = "Mean of the histogram ('Mean')",
             y_label = "Histogram tendency ('Tendency')",
             plot_title = "Relationship between 'Mean' and 'Tendency'") +
  scale_color_manual(values = brewer.pal(n = 3, name = "Set2"),
                     labels = c("left asymmetric", "symmetric",
                                "right asymmetric")) +
  scale_linetype_manual(values = rep(1, 3),
                        labels = c("left asymmetric", "symmetric",
                                   "right asymmetric"))

p

## # Save as PDF:
## ggsave(file="mypathtofolder/FigureXY1.pdf", width=10, height=7)


## 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`

---

`predict.multifor` *Multi forest prediction*

---

## Description

Prediction with new data and a saved forest from `multifor`.

## Usage

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

## Arguments

| | |
|---|---|
| object | multifor object. |
| data | New test data of class data.frame. |
| predict.all | Return individual predictions for each tree instead of aggregated predictions for all trees. Return a matrix (sample x tree) for classification, a 3d array for probability estimation (sample x class x tree). |
| num.trees | Number of trees used for prediction. The first num.trees in the forest are used. |
| type | Type of prediction. If "response" (default), the predicted classes (classification) or predicted probabilities (probability estimation) are returned. If "terminalNodes", the IDs of the terminal node in each tree for each observation in the given dataset are returned. |
| 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.

## Value

Object of class multifor.prediction with elements

| | |
|---|---|
| predictions | Predicted classes/values (only for classification and regression) |
| num.trees | Number of trees. |
| num.independent.variables | Number of independent variables. |
| num.samples | Number of samples. |
| treetype | Type of forest/tree. Classification or probability. |

## 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

multifor

---

`predictions.divfor` *Diversity Forest predictions*

---

## Description

Diversity Forest predictions

## Usage

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

## Arguments

| | |
|---|---|
| `x` | `divfor` 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

divfor

## predictions.divfor.prediction *Diversity Forest predictions*

### Description

Diversity Forest predictions

### Usage

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

### Arguments

| | |
|---|---|
| x | divfor.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

divfor

## 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/ (Accessed: 29/08/2024)

## 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