Package 'unityForest'

May 8, 2026

Type Package

Title Improving Interaction Modelling and Interpretability in Random Forests

Version 0.2.0

Date 2026-02-25

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

Description Implementation of the unity forest (UFO) framework (Hornung & Hapfelmeier, 2026, <doi:10.48550/arXiv.2601.07003>).
UFOs are a random forest variant designed to better take covariates with purely interaction-based effects into account, including interactions for which none of the involved covariates exhibits a marginal effect.
While this framework tends to improve discrimination and predictive accuracy compared to standard random forests, it also facilitates the identification and interpretation of (marginal or interactive) effects: In addition to the UFO algorithm for tree construction, the package includes the unity variable importance measure (unity VIM), which quantifies covariate effects under the conditions in which they are strongest - either marginally or within subgroups defined by interactions - as well as covariate-representative tree roots (CRTRs) that provide interpretable visualizations of these conditions.
Categorical and continuous outcomes are supported.
This package is a fork of the R package 'ranger' (main author: Marvin N. Wright), which implements random forests using an efficient C++ backend.

SystemRequirements C++17

Encoding UTF-8

License GPL-3

Imports Rcpp (>= 0.11.2), Matrix, ggplot2, ggrepel, dplyr, scales, rlang

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.5)

Suggests patchwork

RoxygenNote 7.3.3

NeedsCompilation yes

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

Repository CRAN

Date/Publication 2026-02-25 17:50:02 UTC

Contents

`unityForest-package` *Unity Forest (UFO) Framework*

Description

This package implements the unity forest (UFO) framework. UFOs are a random forest variant designed to better take covariates with purely interaction-based effects into account, including interactions for which none of the involved covariates exhibits a marginal effect. While this framework tends to improve discrimination and predictive accuracy compared to standard random forests, it also facilitates the identification and interpretation of (marginal or interactive) effects: In addition to the UFO algorithm for tree construction, the package includes the unity variable importance measure (unity VIM), which quantifies covariate effects under the conditions in which they are strongest - either marginally or within subgroups defined by interactions - as well as covariate-representative tree roots (CRTRs) that provide interpretable visualizations of these conditions. Categorical and continuous outcomes are supported.

Details

The main functions of the package are:

- `unityfor`: Construct a UFO and compute the unity VIM.
- `predict.unityfor`: Predict using a UFO fitted using `unityfor`.
- `reprTrees`: Select and visualize covariate-representative tree roots (CRTRs) based on a `unityfor` object.

This package is a fork of the R package 'ranger' that implements random forests using an efficient C++ implementation. The documentation is partly taken from 'ranger'.

The code in the example sections can be used as a template for basic application scenarios.

References

- Hornung, R., Hapfelmeier, A. (2026). Unity Forests: Improving Interaction Modelling and Interpretability in Random Forests. arXiv:2601.07003, <doi:10.48550/arXiv.2601.07003>.
- 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>.

`predict.unityfor` *Unity Forest prediction*

Description

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

Usage

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

Arguments

`object`	`unityfor` 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 and continuous outcomes, and 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. Either 'response' (default) or 'terminalNodes'. See below for details.

num.threads	Number of threads to use. The default is to use at most 2 threads (and at most the number of available CPU cores). This conservative default avoids unintentionally using many cores on shared computing resources (e.g., CI systems, servers, or HPC login/compute nodes). For typical use on a personal computer, setting num.threads = 0 is strongly recommended, as it uses all available CPU cores, which typically substantially reduces runtime.
verbose	Verbose output on or off.
...	further arguments passed to or from other methods.

Details

For type = 'response' (the default), the predicted probabilities (probability estimation), predicted classes (classification), or predicted numeric values (continuous outcomes) are returned. For type = 'terminalNodes', the IDs of the terminal node in each tree for each observation in the given dataset are returned.

Value

Object of class unityfor.prediction with elements

predictions	Predicted classes/probabilities/values/terminal nodes
num.trees	Number of trees.
num.independent.variables	Number of independent variables.
treetype	Type of forest/tree. Categorical or continuous outcome.
num.samples	Number of samples.

Author(s)

Marvin N. Wright, Roman Hornung

References

- Hornung, R., Hapfelmeier, A. (2026). Unity Forests: Improving Interaction Modelling and Interpretability in Random Forests. arXiv:2601.07003, <doi:10.48550/arXiv.2601.07003>.
- 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

unityfor

`reprTrees` *Select and visualize covariate-representative tree roots (CRTRs)*

Description

Implements the algorithm for selecting and visualizing covariate-representative tree roots (CRTRs) as described in Hornung & Hapfelmeier (2026).
CRTRs are tree roots extracted from a unity forest that characterize the conditions under which a given variable exhibits its strongest effect on the outcome. The function selects one representative tree root for each variable and visualizes its structure to facilitate interpretation. CRTRs are essential for analyzing the effects identified by the unity VIM (`unityfor`). See the 'Details' section below for more details.

Usage

```
reprTrees(
  object,
  vars = NULL,
  numvars = 5,
  indvars = NULL,
  num.threads = NULL,
  plotit = TRUE,
  highlight_relevant = TRUE,
  box_plots = TRUE,
  density_plots = TRUE,
  scatter_plots = TRUE,
  add_split_line = TRUE,
  verbose = TRUE
)
```

Arguments

`object`	Object of class `unityfor`.
`vars`	This is an optional vector of variable names, for which CRTRs should be obtained
`numvars`	The number of the variables with the largest unity VIM values for which CRTRs should be obtained.
`indvars`	The indices of the variables with the largest unity VIM values for which CRTRs should be obtained. For example, if `indvars = c(1, 3)`, the CRTRs for the variables with the largest and third-largest unity VIM values are obtained.
`num.threads`	Number of threads to use. The default is to use at most 2 threads (and at most the number of available CPU cores). This conservative default avoids unintentionally using many cores on shared computing resources (e.g., CI systems, servers, or HPC login/compute nodes). For typical use on a personal computer, setting `num.threads = 0` is strongly recommended, as it uses all available CPU cores, which typically substantially reduces runtime.

`plotit` Whether or not the CRTRs should be plotted or merely returned (invisibly). Default is `TRUE`.

`highlight_relevant` Whether or not the nodes not containing the top-scoring splits for the variables of interest or their ancestor nodes should be shaded out. Default is `TRUE`. See the 'Details' section below for explanation.

`box_plots` Whether boxplots should be used to show the outcome class-specific distributions of the variables values in the nodes with top-scoring splits (see 'Details' section for explanation). For classification only. Default is `TRUE`.

`density_plots` Whether kernel density plots should be used to show the outcome class-specific distributions of the variable values in the nodes with top-scoring splits (see 'Details' section for explanation). For classification only. Default is `TRUE`.

`scatter_plots` Whether scatter plots should be used to investigate the relationship between the variables of interest and the outcome in the nodes with top-scoring splits (see 'Details' section for explanation). For continuous outcomes only. Default is `TRUE`.

`add_split_line` Whether in the boxplots/density plots/scatter plots a line at the split point of the corresponding node should be drawn. Default is `TRUE`.

`verbose` Verbose output on or off. Default is `TRUE`.

Details

Further details on the descriptions below are provided in Hornung & Hapfelmeier (2026).

Covariate-representative tree roots (CRTRs). Covariate-representative tree roots (CRTRs) (Hornung & Hapfelmeier, 2026) are tree fragments (or 'tree roots' - the first few splits in the trees) extracted from a fitted unity forest (`unityfor`) that characterize for given variables the conditions under which each variable exerts its strongest influence on the prediction.

Technically, for a given variable, the algorithm identifies tree roots in which this variable attains particularly high split scores (top-scoring splits). From these tree roots, a representative root is extracted (Laabs et al., 2024) that best reflects the conditions under which this variable has its strongest effect.

Interpretation and subgroup effects. If a variable has a strong marginal effect, the corresponding CRTR typically contains a split on this variable at the root node (first split in the tree). In contrast, if a variable has little marginal effect but interacts with another variable, the CRTR may first split on that other variable, thereby defining a subgroup in which the variable of interest exhibits a strong conditional effect.

From a substantive perspective, CRTRs enable the exploration of variable effects that are generally not detectable by conventional methods focusing on marginal associations. In particular, CRTRs can reveal variables that have weak marginal effects but act strongly within specific subgroups defined by interactions with other variables.

Relation to unity VIM. CRTRs are closely related to the unity variable importance measure (unity VIM) (`unityfor`). The unity VIM quantifies the strength of variable effects under the conditions in which they are strongest. Analogously, CRTRs visualize these conditions by displaying the tree structures that give rise to the respective unity VIM values.

Accordingly, the CRTR algorithm can be used to visualize and interpret the effects identified by the unity VIM. By default, CRTRs are constructed and visualized for the five variables with the largest unity VIM values.

Scope of applicability. CRTRs should primarily be examined for variables with sufficiently large unity VIM values. Constructing CRTRs for variables with negligible importance may lead to over-interpretation, as apparent patterns may reflect random structure rather than meaningful effects.

Shaded regions in the visualization. For improved interpretability, parts of the CRTRs are shaded out by default. Specifically, only the nodes containing the top-scoring splits for the variable of interest and their ancestor nodes are shown prominently.

This design is motivated by two considerations. First, the purpose of CRTRs is to depict the conditions under which a variable exhibits its strongest effects - conditions that are defined by the ancestors of the nodes with top-scoring splits. Second, the remaining regions of the tree are of limited interpretive value. Since each CRTR is derived from tree roots selected for strong effects of a specific variable, the splitting patterns along the highlighted paths are specific for that variable. In contrast, shaded regions reflect arbitrary aspects of the overall association structure in the data and may include splits on non-informative variables, as each tree root is grown from a (small) random subset of all available variables.

Note that additional splits on the variable of interest may occur within shaded regions and can still be relevant. However, these splits do not represent the conditions under which the variable attains its strongest effects.

In-bag data for top-scoring split visualizations. The boxplots/density plots/scatter plots illustrating the discriminatory power of the top-scoring splits are computed exclusively based on the in-bag observations of the corresponding trees. This is consistent with the construction of the CRTRs themselves, which are derived from in-bag data only.

NOTE: The empirical evaluation of the unity forest framework (including the unity forest algorithm, the unity VIM, and covariate-representative tree roots) in Hornung & Hapfelmeier (2026) focused on categorical outcomes. Its performance for continuous outcomes has not yet been systematically investigated. Results for continuous outcomes should therefore be interpreted with appropriate caution.

Value

Object of class `unityfor.reprTrees` with elements

`rules`	List. In-bag statistics on the outcome at each node in the CRTRs. For classification, this provides the class frequencies and the numbers of observations representing each class.
`plots`	List. Generated ggplot2 plots.
`var.names`	Labels of the variables for which CRTRs were selected.
`independent.variable.names`	Names of all independent variables in the dataset.
`num.independent.variables`	Number of independent variables in the dataset.
`num.samples`	Number of observations in the dataset.
`treetype`	Tree type.
`forest`	Sub-forest that contains only the CRTRs.

Author(s)

Roman Hornung

References

- Hornung, R., Hapfelmeier, A. (2026). Unity Forests: Improving Interaction Modelling and Interpretability in Random Forests. arXiv:2601.07003, <doi:10.48550/arXiv.2601.07003>.
- Laabs, B.-H., Westenberger, A., & König, I. R. (2024). Identification of representative trees in random forests based on a new tree-based distance measure. Advances in Data Analysis and Classification 18(2):363-380, <doi:10.1007/s11634023005377>.

See Also

unityfor

Examples

```
## IMPORTANT NOTE on parallelization:
## The default uses at most 2 threads (num.threads) to avoid unintentionally
## using many cores on shared systems.
## However, for typical runs on a personal computer, set num.threads = 0 to
## use all available CPU cores; this is strongly recommended and can
## substantially reduce runtime.
## Note: num.threads = 1 is used in the examples to avoid parallel
## execution during package checks.


## Load package:

library("unityForest")



## Categorical outcome:
########################

## Set seed to make results reproducible:

set.seed(1234)


## Load wine dataset:

data(wine)


## Construct unity forest and calculate unity VIM values:

model <- unityfor(dependent.variable.name = "C", data = wine,
                  importance = "unity", num.trees = 2000, num.threads = 1)
```

```
# NOTE: num.trees = 2000 (in the above) would be too small for practical
# purposes. This quite small number of trees was simply used to keep the
# runtime of the example short.
# The default number of trees is num.trees = 20000.


## Visualize the CRTRs for the five variables with the largest unity VIM
## values:

reprTrees(model, box_plots = FALSE, density_plots = FALSE,
          num.threads = 1)


## Visualize the CRTRs for the variables with the largest and third-largest
## unity VIM values:

reprTrees(model, indvars = c(2, 3), box_plots = FALSE, density_plots = FALSE,
          num.threads = 1)


## Visualize the CRTRs for the variables with the largest and third-largest
## unity VIM values, where density plots are shown to visualize the
## outcome class-specific distributions of the variables values in the
## nodes with top-scoring splits:

reprTrees(model, indvars = c(2, 3), box_plots = FALSE, density_plots = TRUE,
          num.threads = 1)


## Visualize the CRTRs for the variables with the largest and third-largest
## unity VIM values, where both density plots and boxplots are shown to
## visualize the outcome class-specific distributions of the variables values
## in the top-scoring splits; the split points are not indicated in these
## plots:
ps <- reprTrees(model, indvars = c(2, 3), add_split_line = FALSE,
                num.threads = 1)


## Save one of the CRTRs with the corresponding density plot:

library("patchwork")
library("ggplot2")

p <- ps$plots[[1]]$tree_plot / ps$plots[[1]]$density_plot +
     patchwork::plot_layout(heights = c(2, 1))
p

# outfile <- file.path(tempdir(), "figure_xy.pdf")
# ggsave(outfile, device = cairo_pdf, plot = p, width = 18,
#        height = 14)
```

```
# Note: The plots can be manipulated with the usual ggplot2 syntax, e.g.:

ps$plots[[1]]$density_plot + xlab("Proline") + labs(title = NULL, y = NULL) +
  theme(
    legend.position = c(0.95, 0.95),
    legend.justification = c(1, 1)
  )




## Continuous outcome:
#######################


## Set seed to make results reproducible:

set.seed(1234)


## Load stock dataset:

data(stock)


## Construct unity forest and calculate unity VIM values:

model <- unityfor(dependent.variable.name = "company10", data = stock,
                  importance = "unity", num.trees = 2000, num.threads = 1)

# NOTE: num.trees = 2000 (in the above) would be too small for practical
# purposes. This quite small number of trees was simply used to keep the
# runtime of the example short.
# The default number of trees is num.trees = 20000.


## Visualize the CRTRs for the variables "company1" und "company7", where
## scatter plots are shown to  visualize the effect of the variables in the
## top-scoring splits:

ps <- reprTrees(model, vars = c("company1", "company7"), num.threads = 1)


## Visualize the CRTRs for the variables "company1" and "company7" without
## scatter plots:

reprTrees(model, vars = c("company1", "company7"), scatter_plots = FALSE,
          num.threads = 1)



# As also shown above, the plots can be manipulated with the usual
# ggplot2 syntax, e.g.:
```

```
library("ggplot2")

p <- ps$plots[[1]]$scatter_plot + labs(x = "Stock price of company 1",
                                       y = "Stock price of company 10") +
  ggtitle("Marginal influence of the stock price of company 1")
p

p$layers[[1]]$aes_params$shape <- 1
p$layers[[1]]$aes_params$colour <- "red"
p
```

`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)
```

---

unityfor *Construct a unity forest prediction rule and compute the unity VIM.*

Description

Constructs a unity forest and computes the unity variable importance measure (VIM), as described in Hornung & Hapfelmeier (2026). Categorical and continuous outcomes are supported.
The unity forest algorithm is a tree construction approach for random forests in which the first few splits are optimized jointly in order to more effectively capture interaction effects beyond marginal effects. The unity VIM quantifies the influence of each variable under the conditions in which that influence is strongest, thereby placing a stronger emphasis on interaction effects than conventional variable importance measures.
To explore the nature of the effects identified by the unity VIM, it is essential to examine covariate-representative tree roots (CRTRs), which are implemented in reprTrees.

Usage

```
unityfor(
  formula = NULL,
  dependent.variable.name = NULL,
  data = NULL,
  num.trees = 20000,
  num.cand.trees = 500,
  probability = TRUE,
  importance = "none",
  prop.best.splits = NULL,
  min.node.size.root = NULL,
  min.node.size = NULL,
  max.depth.root = NULL,
  max.depth = NULL,
  prop.var.root = NULL,
  mtry.sprout = NULL,
  replace = FALSE,
  sample.fraction = ifelse(replace, 1, 0.7),
  case.weights = NULL,
  class.weights = NULL,
  inbag = NULL,
  oob.error = TRUE,
  num.threads = NULL,
  write.forest = TRUE,
  verbose = TRUE
)
```

Arguments

formula	Object of class formula or character describing the model to fit. Interaction terms supported only for numerical variables.
dependent.variable.name	Name of the outcome variable, required if no formula is provided. For categorical outcomes, this variable must be coded as a factor.
data	Training data of class data.frame, matrix, dgCMatrix (Matrix) or gwaa.data (GenABEL).
num.trees	Number of trees. Default is 20000.
num.cand.trees	Number of random candidate trees to generate for each tree root. Default is 500.
probability	Grow a probability forest as in Malley et al. (2012). Default is TRUE. For categorical outcomes only.
importance	Variable importance mode, either 'unity' (unity VIM) or 'none'.
prop.best.splits	Related to the unity VIM. Default value should generally not be modified by the user. When calculating the unity VIM, only the top prop.best.splits $\times$ 100% of the splits – those with the highest split criterion values weighted by node size – are considered for each variable. The default value is 0.01, meaning that only the top 1% of splits are used. While small values are recommended, they should not be set too low to ensure that each variable has a sufficient number of splits for a reliable unity VIM computation.
min.node.size.root	Minimal node size in the tree roots. Default is 10 irrespective of the outcome type.
min.node.size	Minimal node size. Default 5 for probability, 5 for classification, and 5 for continuous outcomes.
max.depth.root	Maximal depth of the tree roots. Default value is 3 and should generally not be modified by the user. Larger values can be associated with worse predictive performance for some datasets.
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). Must be at least as large as max.depth.root.
prop.var.root	Proportion of variables randomly sampled for constructing each tree root. Default is the square root of the number of variables divided by the number of variables. Consequently, per default, for each tree root, a random subset of variables is considered, with size equal to the (rounded up) square root of the total number of variables. An exception is made for datasets with more than 100 variables, where the default for prop.var.root is set to 0.1. See the 'Details' section below for explanation.
mtry.sprout	Number of randomly sampled variables to possibly split at in each node of the tree sprouts (i.e., the branches of the trees beyond the tree roots). Default is the (rounded down) square root of the number variables.
replace	Sample with replacement. Default is FALSE.

`sample.fraction`
: Fraction of observations to sample for each tree. Default is 1 for sampling with replacement and 0.7 for sampling without replacement.

`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.

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

`oob.error`
: Compute OOB prediction error. Set to `FALSE` to save computation time.

`num.threads`
: Number of threads to use. The default is to use at most 2 threads (and at most the number of available CPU cores). This conservative default avoids unintentionally using many cores on shared computing resources (e.g., CI systems, servers, or HPC login/compute nodes).

  For typical use on a personal computer, setting `num.threads = 0` is strongly recommended, as it uses all available CPU cores, which typically substantially reduces runtime.

`write.forest`
: Save `unityfor.forest` object, required for prediction. Set to `FALSE` to reduce memory usage if no prediction intended.

`verbose`
: Show computation status and estimated runtime.

Details

There are two reasons why, for datasets with more than 100 variables, the default value of `prop.var.root` is set to 0.1 rather than to the square root of the number of variables divided by the total number of variables.

First, as the total number of variables increases, the square-root-based proportion decreases. This makes it less likely that the same pairs of variables are selected together in multiple trees. This can be problematic for the unity VIM, particularly for variables that do not have marginal effects on their own but act only through interactions with one or a few other variables. Such variables are informative in tree roots only when they are used jointly with the covariates they interact with. Setting `prop.var.root = 0.1` ensures that interacting covariates are selected together sufficiently often in tree roots.

Second, this choice reflects the fact that in high-dimensional datasets, typically only a small proportion of variables are informative. Applying the square-root rule in such settings may result in too few informative variables being selected, thereby reducing the likelihood of constructing predictive tree roots.

However, note that results obtained from applications of the unity forest framework to high-dimensional datasets should be interpreted with caution. For high-dimensional data, the curse of dimensionality makes the identification of individual interaction effects challenging and increases the risk of false positives. Moreover, the split points identified in the CRTRs (`reprTrees`) may become less precise as the number of covariates considered per tree root increases.

NOTE: The empirical evaluation of the unity forest framework (including the unity forest algorithm, the unity VIM, and covariate-representative tree roots) in Hornung & Hapfelmeier (2026) focused

on categorical outcomes. Its performance for continuous outcomes has not been systematically investigated. Results for continuous outcomes should therefore be interpreted with appropriate caution.

Value

Object of class `unityfor` with elements

`predictions`	Predicted classes/class probabilities/values, based on out-of-bag samples.
`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.
`data`	Training data.
`variable.importance`	Variable importance for each independent variable. Only available if `importance` is not `"none"`.
`importance.mode`	Importance mode used.
`prediction.error`	Overall out-of-bag prediction error. For classification this is the fraction of missclassified samples, for probability estimation the Brier score and for continuous outcomes the mean squared error.
`confusion.matrix`	Contingency table for classes and predictions based on out-of-bag samples (classification only).
`r.squared`	R squared. Also called explained variance or coefficient of determination (continuous outcomes only). Computed on out-of-bag data.
`call`	Function call.
`num.trees`	Number of trees.
`num.cand.trees`	Number of candidate trees generated for each tree root.
`num.independent.variables`	Number of independent variables.
`num.samples`	Number of samples.
`prop.var.root`	Proportion of variables randomly sampled for each tree root.
`mtry`	Value of mtry used (in the tree sprouts).
`max.depth.root`	Maximal depth of the tree roots.
`min.node.size.root`	Minimal node size in the tree roots.
`min.node.size`	Value of minimal node size used.
`splitrule`	Splitting rule (used only in the tree sprouts).
`replace`	Sample with replacement.
`treetype`	Type of forest/tree. Categorical or continous outcome.

Author(s)

Roman Hornung, Marvin N. Wright

References

- Hornung, R., Hapfelmeier, A. (2026). Unity Forests: Improving Interaction Modelling and Interpretability in Random Forests. arXiv:2601.07003, <doi:10.48550/arXiv.2601.07003>.
- 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>.

See Also

predict.unityfor

Examples

```
## IMPORTANT NOTE on parallelization:
## The default uses at most 2 threads (num.threads) to avoid unintentionally
## using many cores on shared systems.
## However, for typical runs on a personal computer, set num.threads = 0 to
## use all available CPU cores; this is strongly recommended and can
## substantially reduce runtime.
## Note: num.threads = 1 is used in the examples to avoid parallel
## execution during package checks.


## Load package:

library("unityForest")


## Set seed to make results reproducible:

set.seed(1234)


## Load wine dataset:

data(wine)


## Construct unity forest and calculate unity VIM values:

model <- unityfor(dependent.variable.name = "C", data = wine,
                  importance = "unity", num.trees = 20, num.threads = 1)

# 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.


## Inspect the rankings of the variables and variable pairs with respect to
## the unity VIM:

sort(model$variable.importance, decreasing = TRUE)


## Prediction:

# Separate 'wine' dataset randomly in training
# and test data:
train.idx <- sample(nrow(wine), 2/3 * nrow(wine))
wine_train <- wine[train.idx, ]
wine_test <- wine[-train.idx, ]

# Construct unity forest on training data:
# NOTE again: num.trees = 20 is specified too small for practical purposes.
model_train <- unityfor(dependent.variable.name = "C", data = wine_train,
                        importance = "none", num.trees = 20,
                        probability = FALSE, num.threads = 1)
# NOTE: Because we are only interested in prediction here, we do not
# calculate unity VIM values (by setting importance = "none"), because
# this speeds up calculations.
# Moreover, 'probability' is set to 'FALSE' because we are interested in pure
# class prediction in this example (without class probability prediction).

# Predict class values of the test data:
pred_wine <- predict(model_train, data = wine_test, num.threads = 1)

# Compare predicted and true class values of the test data:
table(wine_test$C, pred_wine$predictions)



## Prediction for dataset with continuous outcome:

# Load stock dataset:

data(stock)

# Separate 'stock' dataset randomly in training
# and test data:
train.idx <- sample(nrow(stock), 2/3 * nrow(stock))
stock_train <- stock[train.idx, ]
stock_test <- stock[-train.idx, ]

# Construct unity forest on training data:
# NOTE again: num.trees = 20 is specified too small for practical purposes.
model_train <- unityfor(dependent.variable.name = "company10",
                        data = stock_train, importance = "none",
                        num.trees = 20, num.threads = 1)
```

```
# NOTE: Because we are only interested in prediction here, we do not
# calculate unity VIM values (by setting importance = "none"), because
# this speeds up calculations.

# Predict outcome values of the test data:
pred_stock <- predict(model_train, data = stock_test, num.threads = 1)

# Compare predicted and true outcome values of the test data:
plot(pred_stock$predictions, stock_test$company10)
```

`wine` *Wine Chemical Analysis Data (Binary Cultivar)*

Description

The well-known wine dataset comprises the results of chemical analyses of 178 wines produced in the same region of Italy from three grape varieties (Barolo, Grignolino, and Barbera). The dataset was originally introduced by Forina et al. (1984) and later described in detail by Forina et al. (1986).

Format

A data frame with 178 observations, 13 numeric covariates and one binary target variable.

Details

For each sample, 13 continuous chemical constituents were measured, which serve as covariates for distinguishing between the grape varieties. For the analyses in this package, a version of the dataset with a binary outcome is provided that differentiates between Grignolino (`"G"`) and the two other varieties (`"Other"`; Barolo and Barbera). This version is available on OpenML under data ID 973.

The variables are as follows:

- `Alc`. numeric. Alcohol.
- `Mal`. numeric. Malic acid.
- `Ash`. numeric. Ash.
- `AlcAsh`. numeric. Alkalinity of ash.
- `Mg`. numeric. Magnesium.
- `TP`. numeric. Total phenols.
- `Fla`. numeric. Flavonoids.
- `NFP`. numeric. Nonflavonoid phenols.
- `ProAn`. numeric. Proanthocyanins.
- `Col`. numeric. Color intensity.
- `Hue`. numeric. Hue.
- `WAI`. numeric. OD280/OD315 of diluted wines (wine absorbance index).
- `Prol`. numeric. Proline.
- `C`. factor. Cultivar. Binary target variable: `"G"` vs `"Other"`.

Source

OpenML: data.id: 973, link: https://www.openml.org/d/973/

References

- Forina, M. (1984). PARVUS, TrAC Trends in Analytical Chemistry, 3(2):38–39, <doi:10.1016/01659936(84)870508>.
- Forina, M., Armanino, C., Castino, M., Ubigli, M. (1986). Multivariate data analysis as a discriminating method of the origin of wines, Vitis, 25:189–201, <doi:10.5073/vitis.1986.25.189-201>.
- 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

```
data(wine)

table(wine$C)
dim(wine)

head(wine)
```

Index