Package ‘moreparty’

October 13, 2022

Type Package

Title A Toolbox for Conditional Inference Trees and Random Forests

Version 0.3

Depends R (>= 3.5.0), party

Imports partykit, varImp, plyr, foreach, measures, methods, MASS, iml, pdp, vip, ggplot2, rlang, shiny, shinyWidgets, rclipboard, DT, datamods, phosphoricons

Suggests doParallel, knitr, rmarkdown, rmdformats, GDAtools, RColorBrewer, caret, pROC, tidyverse, e1071

VignetteBuilder knitr

Author Nicolas Robette

Maintainer Nicolas Robette <nicolas.robette@uvsq.fr>

Description Additions to 'party' and 'partykit' packages : tools for the interpretation of forests (surrogate trees, prototypes, etc.), feature selection (see Gregorutti et al (2017) <arXiv:1310.5726>, Hapfelmeier and Ulm (2013) <doi:10.1016/j.csda.2012.09.020>, Altmann et al (2010) <doi:10.1093/bioinformatics/btq134>) and parallelized versions of conditional forest and variable importance functions. Also modules and a shiny app for conditional inference trees.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-06-15 17:00:02 UTC

R topics documented:

`BivariateAssoc` *Bivariate association measures for supervised learning tasks.*

Description

Computes bivariate association measures between a response and predictor variables (and, optionnaly, between every pairs of predictor variables.)

Usage

```
BivariateAssoc(Y, X, xx = TRUE)
```

Arguments

`Y`	the response variable
`X`	the predictor variables
`xx`	whether the association measures should be computed for couples of predictor variables (default) or not. With a lot of predictors, consider setting xx to FALSE (for reasons of computation time).

Details

For each pair of variable, a permutation test is computed, following the framework used in conditional inference trees to choose a splitting variable. This test produces a p-value, transformed as -log(1-p) for reasons of comparison stability. The function also computes a "standard" association measure : kenddal's tau correlation for pairs of numeric variables, Cramer's V for pairs of factors and eta-squared for pairs numeric-factor.

Value

A list of the following items :

- `YX` : a table with the association measures between the response and predictor variables
- `XX` : a table with the association measures between every couples of predictor variables

In each table :

- `measure` : name of the "standard" association measure
- `assoc` : value of the "standard" association measure
- `p.value` : p-value from the permutation test
- `criterion` : p-value from the permutation test transformed as -log(1-p), which serves to sort rows

Note

see also https://stats.stackexchange.com/questions/171301/interpreting-ctree-partykit-output-in-r

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

`ctree`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
BivariateAssoc(iris2$Species,iris2[,1:4])
```

ctree-module *Shiny module to build and analyse conditional inference trees*

Description

The module builds a conditional inference trees according to several parameter inputs. Then it plots the tree and computes performance measures, variable importance, checks the stability and return the code to reproduce the analyses.

Usage

```
ctreeUI(id)

ctreeServer(id, data, name)
```

Arguments

`id`	Module id. See `shiny::callModule()`.
`data`	`shiny::reactive()` function returning a `data.frame` to use for the analyses.
`name`	`shiny::reactive()` function returning a `character` string representing `data` name.

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

`ictree`

Examples

```
library(shiny)
library(moreparty)

data(titanic)

ui <- fluidPage(
  titlePanel("Conditional inference trees"),
  ctreeUI(id = "ctree_app")
```

```
)

server <- function(input, output, session) {
  rv <- reactiveValues(
    data = titanic,
    name = deparse(substitute(titanic))
  )
  ctreeServer(id = "ctree_app", reactive(rv$data), reactive(rv$name))
}

if (interactive())
  shinyApp(ui, server)
```

`EasyTreeVarImp` *Variable importance for conditional inference trees.*

Description

Variable importance for `partykit` conditional inference trees, using various performance measures.

Usage

```
EasyTreeVarImp(ct, nsim = 1)
```

Arguments

`ct`	A tree of class `constparty` (as returned by `ctree` from `partykit` package).
`nsim`	Integer specifying the number of Monte Carlo replications to perform. Default is 1. If nsim > 1, the results from each replication are simply averaged together.

Details

If the response variable is a factor, AUC (if response is binary), accuracy, balanced accuracy and true predictions by class are used. If the response is numeric, r-squared and Kendall's tau are used.

Value

A data frame of variable importances, with variables as rows and performance measures as columns.

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

ctree

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.ct = partykit::ctree(Species ~ ., data = iris2)
EasyTreeVarImp(iris.ct, nsim = 1)
```

---

fastcforest — *Parallelized conditional inference random forest*

Description

Parallelized version of cforest function from party package, which is an implementation of the random forest and bagging ensemble algorithms utilizing conditional inference trees as base learners.

Usage

```
fastcforest(formula, data = list(), subset = NULL, weights = NULL,
            controls = party::cforest_unbiased(),
            xtrafo = ptrafo, ytrafo = ptrafo, scores = NULL,
            parallel = TRUE)
```

Arguments

formula	a symbolic description of the model to be fit. Note that symbols like : and - will not work and the tree will make use of all variables listed on the rhs of formula
data	a data frame containing the variables in the model
subset	an optional vector specifying a subset of observations to be used in the fitting process
weights	an optional vector of weights to be used in the fitting process. Non-negative integer valued weights are allowed as well as non-negative real weights. Observations are sampled (with or without replacement) according to probabilities weights / sum(weights). The fraction of observations to be sampled (without replacement) is computed based on the sum of the weights if all weights are integer-valued and based on the number of weights greater zero else. Alternatively, weights can be a double matrix defining case weights for all ncol(weights) trees in the forest directly. This requires more storage but gives the user more control.
controls	an object of class ForestControl-class, which can be obtained using cforest_control (and its convenience interfaces cforest_unbiased and cforest_classical).

xtrafo	a function to be applied to all input variables. By default, the ptrafo function is applied.
ytrafo	a function to be applied to all response variables. By default, the ptrafo function is applied.
scores	an optional named list of scores to be attached to ordered factors
parallel	Logical indicating whether or not to run fastcforest in parallel using a backend provided by the foreach package. Default is TRUE.

Details

See cforest documentation for details. The code for parallelization is inspired by https://stackoverflow.com/questions/36272816/train-a-cforest-in-parallel

Value

An object of class RandomForest-class.

Author(s)

Nicolas Robette

References

Leo Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5–32.

Torsten Hothorn, Berthold Lausen, Axel Benner and Martin Radespiel-Troeger (2004). Bagging Survival Trees. *Statistics in Medicine*, **23**(1), 77–91.

Torsten Hothorn, Peter Buhlmann, Sandrine Dudoit, Annette Molinaro and Mark J. van der Laan (2006a). Survival Ensembles. *Biostatistics*, **7**(3), 355–373.

Torsten Hothorn, Kurt Hornik and Achim Zeileis (2006b). Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. Preprint available from https://www.zeileis.org/papers/Hothorn+Hornik+Zeileis-2006.pdf

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis and Torsten Hothorn (2007). Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution. *BMC Bioinformatics*, **8**, 25. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-25

Carolin Strobl, James Malley and Gerhard Tutz (2009). An Introduction to Recursive Partitioning: Rationale, Application, and Characteristics of Classification and Regression Trees, Bagging, and Random forests. *Psychological Methods*, **14**(4), 323–348.

See Also

cforest, fastvarImp

Examples

```
## classification
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species=="versicolor")
iris.cf = fastcforest(Species~., data=iris2, parallel=FALSE)
```

`fastvarImp` *Variable importance for conditional inference random forests*

Description

Parallelized version of `varImp` function from `varImp` package, which computes the variable importance for arbitrary measures from the `measures` package.

Usage

```
fastvarImp(object, mincriterion = 0, conditional = FALSE,
          threshold = 0.2, nperm = 1, OOB = TRUE,
          pre1.0_0 = conditional, measure = "multiclass.Brier",
          parallel = TRUE, ...)
```

Arguments

`object`	An object as returned by `cforest` (or `fastcforest`).
`mincriterion`	The value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default mincriterion = 0 guarantees that all splits are included.
`conditional`	a logical determining whether unconditional or conditional computation of the importance is performed.
`threshold`	The threshold value for (1 - p-value) of the association between the variable of interest and a covariate, which must be exceeded inorder to include the covariate in the conditioning scheme for the variable of interest (only relevant if conditional = TRUE). A threshold value of zero includes all covariates.
`nperm`	The number of permutations performed.
`OOB`	A logical determining whether the importance is computed from the out-of-bag sample or the learning sample (not suggested).
`pre1.0_0`	Prior to party version 1.0-0, the actual data values were permuted according to the original permutation importance suggested by Breiman (2001). Now the assignments to child nodes of splits in the variable of interest are permuted as described by Hapfelmeier et al. (2012), which allows for missing values in the explanatory variables and is more efficient wrt memory consumption and computing time. This method does not apply to conditional variable importances.
`measure`	The name of the measure of the `measures` package that should be used for the variable importance calculation.

`parallel` Logical indicating whether or not to run `fastvarImp` in parallel using a backend provided by the `foreach` package. Default is `FALSE`.

`...` Further arguments (like positive or negative class) that are needed by the measure.

Details

The code is adapted from `varImp` function in `varImp` package.

Value

Vector with computed permutation importance for each variable.

Author(s)

Nicolas Robette

See Also

`varImp`, `fastvarImpAUC`, `cforest`, `fastcforest`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
fastvarImp(object = iris.cf, measure='ACC', parallel=FALSE)
```

`fastvarImpAUC` *Variable importance (with AUC performance measure) for conditional inference random forests*

Description

Computes the variable importance regarding the AUC. Bindings are not taken into account in the AUC definition as they did not provide as good results as the version without bindings in the paper of Janitza *et al.* (2013).

Usage

```
fastvarImpAUC(object, mincriterion = 0, conditional = FALSE,
          threshold = 0.2, nperm = 1, OOB = TRUE,
          pre1.0_0 = conditional,
          parallel = TRUE)
```

Arguments

- `object` An object as returned by `cforest` (or `fastcforest`).
- `mincriterion` The value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default mincriterion = 0 guarantees that all splits are included.
- `conditional` The value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default mincriterion = 0 guarantees that all splits are included.
- `threshold` The threshold value for (1 - p-value) of the association between the variable of interest and a covariate, which must be exceeded inorder to include the covariate in the conditioning scheme for the variable of interest (only relevant if conditional = TRUE). A threshold value of zero includes all covariates.
- `nperm` The number of permutations performed.
- `OOB` A logical determining whether the importance is computed from the out-of-bag sample or the learning sample (not suggested).
- `pre1.0_0` Prior to party version 1.0-0, the actual data values were permuted according to the original permutation importance suggested by Breiman (2001). Now the assignments to child nodes of splits in the variable of interest are permuted as described by Hapfelmeier et al. (2012), which allows for missing values in the explanatory variables and is more efficient wrt memory consumption and computing time. This method does not apply to conditional variable importances.
- `parallel` Logical indicating whether or not to run `fastvarImpAUC` in parallel using a backend provided by the `foreach` package. Default is `FALSE`.

Details

For using the original AUC definition and multiclass AUC you can use the `fastvarImp` function and specify the particular measure. The code is adapted from `varImpAUC` function in `varImp` package.

Value

Vector with computed permutation importance for each variable.

Author(s)

Nicolas Robette

References

Janitza, S., Strobl, C. & Boulesteix, A.-L. An AUC-based permutation variable importance measure for random forests. *BMC Bioinform.* 14, 119 (2013).

See Also

`varImpAUC`, `fastvarImp`, `cforest`, `fastcforest`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
fastvarImpAUC(object = iris.cf, parallel = FALSE)
```

`FeatureSelection` *Feature selection for conditional random forests.*

Description

Performs feature selection for a conditional random forest model. Four approaches are available : non-recursive feature elimination (NRFE), recursive feature elimination (RFE), permutation test approach with permuted response (Altmann et al, 2010), permutation test approach with permuted predictors (Hapfelmeier et Ulm, 2013).

Usage

```
FeatureSelection(Y, X, method = 'NRFE', ntree = 1000, measure = NULL,
                 nperm = 30, alpha = 0.05, distrib = 'approx',
                 parallel = FALSE, ...)
```

Arguments

`Y`	response vector. Must be of class `factor` or `numeric`
`X`	matrix or data frame containing the predictors
`method`	method for feature selection. Should be 'NRFE' (non-recursive feature elimination, default), 'RFE' (recursive feature elimination), 'ALT' (permutation of response) or 'HAPF' (permutation of predictors)
`ntree`	number of trees contained in a forest
`measure`	the name of the measure of the `measures` package that should be used for error and variable importance calculations.
`nperm`	number of permutations. Only for 'ALT' and 'HAPF' methods.
`alpha`	alpha level for permutation tests. Only for 'ALT' and 'HAPF' methods.
`distrib`	the null distribution of the variable importance can be approximated by its asymptotic distribution (`"asympt"`) or via Monte Carlo resampling (`"approx"`, default). Only for 'ALT' and 'HAPF' methods.
`parallel`	Logical indicating whether or not to run `fastvarImp` in parallel using a backend provided by the `foreach` package. Default is `FALSE`.
`...`	Further arguments (like positive or negative class) that are needed by the measure.

Details

To be developed soon !

Value

A list with the following elements :

- `selection.0se` selected variables with the 0 standard error rule
- `forest.0se` forest corresponding the variables selected with the 0 standard error rule
- `oob.error.0se` OOB error of the forest with 0 standard error rule
- `selection.1se` selected variables with the 1 standard error rule
- `forest.1se` forest corresponding the variables selected with the 1 standard error rule
- `oob.error.1se` OOB error of the forest with 1 standard error rule

Note

The code is adapted from Hapfelmeier & Ulm (2013).

Only works for regression and binary classification.

Author(s)

Nicolas Robette

References

B. Gregorutti, B. Michel, and P. Saint Pierre. "Correlation and variable importance in random forests". arXiv:1310.5726, 2017.

A. Hapfelmeier and K. Ulm. "A new variable selection approach using random forests". *Computational Statistics and Data Analysis*, 60:50–69, 2013.

A. Altmann, L. Toloşi, O. Sander et T. Lengauer. "Permutation importance: a corrected feature importance measure". *Bioinformatics*, 26(10):1340-1347, 2010.

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
featsel <- FeatureSelection(iris2$Species, iris2[,1:4], measure='ACC', ntree=200)
featsel$selection.0se
featsel$selection.1se
```

`GetAleData` *Accumulated Local Effects for a conditional random forest.*

Description

Computes the Accumulated Local Effects for several covariates in a conditional random forest and gathers them into a single data frame.

Usage

```
GetAleData(object, xnames=NULL, order=1, grid.size=20, parallel=FALSE)
```

Arguments

`object`	An object as returned by `cforest` (or `fastcforest`).
`xnames`	A character vector of the covariates for which to compute the Accumulated Local Effects. If NULL (default), ALE are computed for all the covariates in the model. Should be of length 2 for 2nd order ALE.
`order`	An integer indicating whether to compute 1st order ALE (1, default) or 2nd order ALE (2).
`grid.size`	The size of the grid for evaluating the predictions. Default is 20.
`parallel`	Logical indicating whether or not to run the function in parallel using a backend provided by the `foreach` package. Default is `FALSE`.

Details

The computation of Accumulated Local Effects uses `FeatureEffect` function from `iml` package for each covariate. The results are then gathered and reshaped into a friendly data frame format.

Value

A data frame with covariates, their categories and their accumulated local effects.

Author(s)

Nicolas Robette

References

Apley, D. W., Zhu J. "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models". arXiv:1612.08468v2, 2019.

Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

See Also

`FeatureEffect`,`GetPartialData`,`GetInteractionStrength`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
            controls = party::cforest_unbiased(mtry=2, ntree=50))
GetAleData(iris.cf)
```

GetCtree *Gets a tree from a conditional random forest*

Description

This function gets the ith tree from a conditional random forest as produced by `cforest`.

Usage

```
GetCtree(object, k = 1)
```

Arguments

`object`	An object as returned by `cforest` (or `fastcforest`).
`k`	The index of the tree to get from the forest. Default is 1.

Value

A tree of class `BinaryTree`, as returned by `ctree` from `party` package.

Note

Code taken from https://stackoverflow.com/questions/19924402/cforest-prints-empty-tree

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
plot(GetCtree(iris.cf))
```

GetInteractionStrength

Strength of interactions

Description

Computes the strength of second order interactions for covariates in a conditional random forest.

Usage

```
GetInteractionStrength(object, xnames=NULL)
```

Arguments

object	An object as returned by cforest (or fastcforest).
xnames	character vector. The names of the variables for which to measure the strength of second order interactions. If NULL (default), all covariates are included.

Value

A data frame with pairs of variable names and the strength of the interaction between them.

Note

This function calls vint function in vip package for each interaction. The results are then gathered and reshaped into a friendly data frame format.

Author(s)

Nicolas Robette

References

Greenwell, B. M., Boehmke, B. C., and McCarthy, A. J.: A Simple and Effective Model-Based Variable Importance Measure. arXiv preprint arXiv:1805.04755 (2018).

See Also

GetPartialData,GetAleData

Examples

```
## Not run:
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
            controls = party::cforest_unbiased(mtry=2, ntree=50))
```

```
  GetInteractionStrength(iris.cf)

## End(Not run)
```

GetPartialData *Partial dependence for a conditional random forest.*

Description

Computes the partial dependence for several covariates in a conditional random forest and gathers them into a single data frame.

Usage

```
GetPartialData(object, xnames=NULL, ice = FALSE, center = FALSE,
               grid.resolution = NULL, quantiles = TRUE, probs = 1:9/10,
               trim.outliers = FALSE, which.class = 1L, prob = TRUE,
               pred.fun = NULL, parallel = FALSE, paropts = NULL)
```

Arguments

object	An object as returned by `cforest` (or `fastcforest`).
xnames	A character vector of the covariates for which to compute the partial dependence. If NULL (default), partial dependence is computed for all the covariates in the model.
ice	Logical indicating whether or not to compute individual conditional expectation (ICE) curves. Default is FALSE. See Goldstein et al. (2014) for details.
center	Logical indicating whether or not to produce centered ICE curves (c-ICE curves). Only used when ice = TRUE. Default is FALSE. See Goldstein et al. (2014) for details.
grid.resolution	Integer giving the number of equally spaced points to use for the continuous variables listed in `xnames`. If left NULL, it will default to the minimum between 51 and the number of unique data points for each of the continuous independent variables listed in `xnames`.
quantiles	Logical indicating whether or not to use the sample quantiles of the continuous predictors listed in `xnames`. If `quantiles = TRUE` and `grid.resolution = NULL` (default), the sample quantiles will be used to generate the grid of joint values for which the partial dependence is computed.
probs	Numeric vector of probabilities with values in [0,1]. (Values up to 2e-14 outside that range are accepted and moved to the nearby endpoint.) Default is `1:9/10` which corresponds to the deciles of the predictor variables. These specify which quantiles to use for the continuous predictors listed in `xnames` when `quantiles = TRUE`.

`trim.outliers`	Logical indicating whether or not to trim off outliers from the continuous predictors listed in `xnames` (using the simple boxplot method) before generating the grid of joint values for which the partial dependence is computed. Default is FALSE.
`which.class`	Integer specifying which column of the matrix of predicted probabilities to use as the "focus" class. Default is to use the first class. Only used for classification problems.
`prob`	Logical indicating whether or not partial dependence for classification problems should be returned on the probability scale, rather than the centered logit. If FALSE, the partial dependence function is on a scale similar to the logit. Default is TRUE.
`pred.fun`	Optional prediction function that requires two arguments: `object` and `newdata`. If specified, then the function must return a single prediction or a vector of predictions (i.e., not a matrix or data frame). Default is NULL.
`parallel`	Logical indicating whether or not to run `partial` in parallel using a backend provided by the `foreach` package. Default is FALSE.
`paropts`	List containing additional options to be passed onto `foreach` when `parallel = TRUE`.

Details

The computation of partial dependence uses `partial` function from pdp package for each covariate. The results are then gathered and reshaped into a friendly data frame format.

Value

A data frame with covariates, their categories and their partial dependence effects.

Author(s)

Nicolas Robette

References

J. H. Friedman. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29: 1189-1232, 2001.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E., Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. (2014) Journal of Computational and Graphical Statistics, 24(1): 44-65, 2015.

See Also

`partial`,`GetAleData`,`GetInteractionStrength`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
            controls = party::cforest_unbiased(mtry=2, ntree=50))
GetPartialData(iris.cf)
```

`GetSplitStats` *Permutation tests results for each split in a conditional tree.*

Description

This function displays the results the selection variable process for each split of a conditional tree, i.e. the p-values from permutation tests of independence between every predictor and the dependent variable. This may help to assess the stability of the tree.

Usage

```
GetSplitStats(ct)
```

Arguments

`ct`	A tree of class `BinaryTree` (as returned by `ctree` from `party` package) or `constparty` (as returned by `ctree` from `partykit` package).

Value

A list of elements, one for each split in the tree. For each split, the vector corresponds to are log(1-p) for every predictors, with p the p-value of the permutation test of independence. Variables are sorted by decreasing degree of association with the dependent variable.

Note

see also https://stats.stackexchange.com/questions/171301/interpreting-ctree-partykit-output-in-r

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

ctree

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.ct = partykit::ctree(Species ~ ., data = iris2)
GetSplitStats(iris.ct)
```

---

ggForestEffects *Dot plot of covariates effects*

Description

Plots the effects (partial dependence or accumulated local effects) of the covariates of a supervised learning model in a single a dot plot.

Usage

```
ggForestEffects(dt, vline=0, xlabel="", ylabel="", main="")
```

Arguments

dt	data frame. Must have three columns : one with the names of the covariates (named "var"), one with the names of the categories of the covariates (named "cat"), one with the values of the effects (named "value"). Typically the result of GetAleData or GetPartialData functions.
vline	numeric. Coordinate on the x axis where a vertical line is added.
xlabel	character. Title of the x axis.
ylabel	character. Title of the y axis.
main	character. Title of the plot.

Note

There should be no duplicated categories. If it is the case, duplicated categories have to be renamed in `dt` prior to running `ggForestEffects`.

Author(s)

Nicolas Robette

References

Apley, D. W., Zhu J. "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models". arXiv:1612.08468v2, 2019.

Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

See Also

GetAleData, GetPartialData

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2, controls = cforest_unbiased(mtry=2))
ale <- GetAleData(iris.cf)
ale$cat <- paste(ale$var,ale$cat,sep='_')  # to avoid duplicated categories
ggForestEffects(ale)
```

---

ggVarImp — *Dot plot of variable importance*

Description

Plots the importance of the covariates of a supervised learning model in a dot plot.

Usage

```
ggVarImp(importance, sort=TRUE, xlabel="Importance", ylabel="Variable", main="")
```

Arguments

`importance`	numeric vector. The vector of the importances of the covariates. Should be a named vector.
`sort`	logical. Whether the vector of importances should be sorted or not. Default is TRUE.
`xlabel`	character. Title of the x axis.
`ylabel`	character. Title of the y axis.
`main`	character. Title of the plot.

Author(s)

Nicolas Robette

See Also

varImp,varImpAUC,fastvarImp,fastvarImpAUC

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
imp <- fastvarImpAUC(object = iris.cf, parallel = FALSE)
ggVarImp(imp)
```

---

ictree *An interactive app for conditional inference trees*

Description

This function launches a shiny app in a web browser in order to build and analyse conditional inference trees.

Usage

```
ictree(treedata = NULL)
```

Arguments

treedata	The data frame to be used in the app. If NULL (default), a module is launched to import data from a file or from the global environment.

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

ctree-module

Examples

```
if (interactive()) {
ictree(iris)
}
```

`NiceTreePlot` *Plots conditional inference trees.*

Description

Plots a `partykit` conditional inference tree in a pretty and simple way.

Usage

```
NiceTreePlot(ct, inner_plots = FALSE)
```

Arguments

`ct`	A tree of class `constparty` (as returned by `ctree` from `partykit` package).
`inner_plots`	Logical. If TRUE, plots are displayed at each inner node. Default is FALSE.

Author(s)

Nicolas Robette

References

Hothorn T, Hornik K, Van De Wiel MA, Zeileis A. "A lego system for conditional inference". *The American Statistician*. 60:257–263, 2006.

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Framework". *Journal of Computational and Graphical Statistics*, 15(3):651-674, 2006.

See Also

`ctree`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.ct = partykit::ctree(Species ~ ., data = iris2)
NiceTreePlot(iris.ct, inner_plots = TRUE)
```

Outliers *Computes outliers*

Description

Computes outlierness scores and detects outliers.

Usage

```
Outliers(prox, cls=NULL, data=NULL, threshold=10)
```

Arguments

prox	a proximity matrix (a square matrix with 1 on the diagonal and values between 0 and 1 in the off-diagonal positions).
cls	Factor. The classes the rows in the proximity matrix belong to. If NULL (default), all data are assumed to come from the same class.
data	A data frame of variables to describe the outliers (optional).
threshold	Numeric. The value of outlierness above which an observation is considered an outlier. Default is 10.

Details

The outlierness score of a case is computed as n / sum(squared proximity), normalized by subtracting the median and divided by the MAD, within each class.

Value

A list with the following elements :

scores	numeric vector containing the outlierness scores
outliers	numeric vector of indexes of the outliers, or a data frame with the outliers and their characteristics

Note

The code is adapted from `outlier` function in `randomForest` package.

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
prox=proximity(iris.cf)
Outliers(prox, iris2$Species, iris2[,1:4])
```

`Prototypes` *Prototypes of groups*

Description

Prototypes are 'representative' cases of a group of data points, given the similarity matrix among the points. They are very similar to medoids.

Usage

```
Prototypes(label, x, prox, nProto = 5, nNbr = floor((min(table(label)) - 1)/nProto))
```

Arguments

`label`	the response variable. Should be a factor.
`x`	matrix or data frame of predictor variables.
`prox`	the proximity (or similarity) matrix, assumed to be symmetric with 1 on the diagonal and in [0, 1] off the diagonal (the order of row/column must match that of x)
`nProto`	number of prototypes to compute for each value of the response variables.
`nNbr`	number of nearest neighbors used to find the prototypes.

Details

For each case in x, the nNbr nearest neighors are found. Then, for each class, the case that has most neighbors of that class is identified. The prototype for that class is then the medoid of these neighbors (coordinate-wise medians for numerical variables and modes for categorical variables). One then remove the neighbors used and iterate the first steps to find a second prototype, etc.

Value

A list of data frames with prototypes. The number of data frames is equal to the number of classes of the response variable.

Note

The code is an extension of `classCenter` function in `randomForest` package.

Author(s)

Nicolas Robette

References

Random Forests, by Leo Breiman and Adele Cutler https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#p

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
prox=proximity(iris.cf)
Prototypes(iris2$Species,iris2[,1:4],prox)
```

`SurrogateTree` *Surrogate tree for conditional inference random forests*

Description

Builds a surrogate tree to approximate a conditional random forest model.

Usage

```
SurrogateTree(object, mincriterion = 0.95, maxdepth = 3)
```

Arguments

`object`	An object as returned by `cforest` (or `fastcforest`).
`mincriterion`	the value of the test statistic (for `testtype == "Teststatistic"`), or 1 - p-value (for other values of `testtype`) that must be exceeded in order to implement a split.
`maxdepth`	maximum depth of the tree. Default is 3.

Details

A global surrogate model is an interpretable model that is trained to approximate the predictions of a black box model (see Molnar 2019). Here a conditional inference tree is build to approximate the prediction of a conditional inference random forest. Practically, the surrogate tree takes the forest predictions as response and the same predictors as the forest.

Value

A list withe following items :

`tree`	The surrogate tree, of class `party`
`r.squared`	The R squared of a linear regression with random forests prediction as dependent variable and surrogate tree prediction as predictor

Note

The surrogate tree is built using `ctree` from `partykit` package.

Author(s)

Nicolas Robette

References

Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

See Also

`cforest`, `ctree`

Examples

```
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2,
          control = party::cforest_unbiased(mtry = 2, ntree = 50))
surro <- SurrogateTree(iris.cf)
surro$r.squared
plot(surro$tree)
```

`titanic` *Titanic dataset*

Description

A dataset describing the passengers of the Titanic and their survival

Usage

```
data("titanic")
```

Format

A data frame with 1309 observations and the following 5 variables.

- `Survived` Factor. Whether one survived or not
- `Pclass` Factor. Passenger class
- `Sex` Factor. Sex
- `Age` Numeric vector. Age
- `Embarked` Factor. Port of embarkation

Examples

```
data(titanic)
str(titanic)
```

Index