Package ‘moreparty’

November 22, 2023

Type Package

Title A Toolbox for Conditional Inference Trees and Random Forests

Version 0.4

Depends R (>= 3.5.0), party

Imports partykit, varImp, plyr, foreach, measures, methods, MASS, iml,
   pdp, vip (>= 0.4.1), ggplot2, rlang, shiny, shinyWidgets,
   rclipboard, DT, datamods, phosphoricons

Suggests doParallel, knitr, rmarkdown, rmdformats, descriptio,
   RColorBrewer, caret, pROC, dplyr, 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 Gre-
forest and variable importance functions. Also modules and a shiny app for conditional in-
ference trees.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2023-11-22 14:30:02 UTC

R topics documented:

   BivariateAssoc .......................................................... 2
ctree-module ............................................................... 4
BivariateAssoc

Bivariate association measures for supervised learning tasks.

Description

Computes bivariate association measures between a response and predictor variables (and, optionally, 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: kendall'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


See Also

ctree

Examples

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

Hothorn T, Hornik K, Zeileis A. "Unbiased Recursive Partitioning: A Conditional Inference Frame-

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 \texttt{partykit} conditional inference trees, using various performance measures.

Usage

\texttt{EasyTreeVarImp(ct, nsim = 1)}

Arguments

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

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


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)

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 back-end 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


See Also

cforest, fastvarImp
Examples

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

Description

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

Usage

```r
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**

```r
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**

```r
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


See Also

varImpAUC, fastvarImp, cforest, fastcforest
FeatureSelection

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


Examples
```r
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```
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


See Also

`FeatureEffect`, `GetPartialData`, `GetInteractionStrength`
GetCtree

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


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))
GetAleData(iris.cf)

## End(Not run)
GetInteractionStrength

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 from an old version of `vip` package for each interaction. The results are then gathered and reshaped into a friendly data frame format.

Author(s)
Nicolas Robette

References

See Also
GetPartialData, GetAleData

Examples
```r
## 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))
```
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


See Also

partial, GetAleData, GetInteractionStrength
Examples

```r
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)
```

Description

This function displays the results of the variable selection 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

cr A tree of class constparty (as returned by ctree from partykit package).

Details

The ratio index represents the ratio between the association test result for the splitting variable and the association test result for another candidate variable for splitting. It is always greater than 1. The closer it is to 1, the tighter the competition for the splitting variable, and therefore the more potentially unstable the node concerned. Conversely, the higher the ratio, the more the splitting variable has dominated the competition, and the more stable the node is likely to be.

Value

A list of two elements:

- `details` a list of data frames (one for each inner node), with one row per candidate variable, and test statistic and p-value of the permutation test of independence, criterion (equal to log(1-p)) and ratio (criterion/max(criterion)) as columns. Variables are sorted by decreasing degree of association with the dependent variable.

- `summary` a data frame with one row per inner node and 5 variables: the mode id, the splitting variable, the best candidate to split among the other variables, the ratio of the criterion of the splitting variable divided by the criterion of the best variable among the others.

Note

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

Author(s)

Nicolas Robette

References


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 `df` prior to running `ggForestEffects`.

Author(s)

Nicolas Robette

References


See Also

`GetAleData`, `GetPartialData`

Examples

```r
## Not run:
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.cf = party::cforest(Species ~ ., data = iris2, controls = cforest_unbiased(mtry=2))
aple <- GetAleData(iris.cf)
aple$cat <- paste(aple$var, aple$cat, sep="_") # to avoid duplicated categories
ggForestEffects(aple)
## End(Not run)
```

---

**ggVarImp**

*Dot plot of variable importance*

Description

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

Usage

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

```r
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

---

**ictree** 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**

```r
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)
```
References


See Also

- `ctree-module`

Examples

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

---------

**NiceTreePlot**

Plots conditional inference trees.

Description

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

Usage

```r
NiceTreePlot(ct, inner_plots = FALSE, cex = 0.8, justmin = 15)
```

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.
- **cex**: Numerical value. Multiplier applied to fontsize. Default is 0.8.
- **justmin**: Numerical value. Minimum average edge label length to employ justification (see `panelfunctions` documentation from partykit package)

Author(s)

Nicolas Robette

References


NodesInfo

See Also
ctree

Examples

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

<table>
<thead>
<tr>
<th>NodesInfo</th>
<th>Informations about terminal nodes</th>
</tr>
</thead>
</table>

Description

Retrieves informations about terminal nodes of a conditional inference tree: node id, rule set, frequency, prediction or class probabilities.

Usage

```r
NodesInfo(ct)
```

Arguments

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

Value

A data frame.

Author(s)

Nicolas Robette

References


See Also
ctree
Examples

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

---

**NodeTreePlot**

Plots the results of each node of a conditional inference tree

---

**Description**

Plots the results of each node of a partykit conditional inference tree with boxplots (regression) or lollipops (binary classification).

**Usage**

```r
NodeTreePlot(ct)
```

**Arguments**

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

**Value**

A ggplot2 object.

**Author(s)**

Nicolas Robette

**References**


**See Also**

ctree

**Examples**

```r
data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.ct = partykit::ctree(Species ~ ., data = iris2)
NodeTreePlot(iris.ct)
```
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])
PerfsBinClassif  
*Performance measures for binary classification tasks*

**Description**

Computes various performance measures for binary classification tasks: true positive rate, true negative rate, accuracy, balanced accuracy, area under curve (AUC).

**Usage**

```r
PerfsBinClassif(pred, actual)
```

**Arguments**

- `pred`: numerical vector of predicted values
- `actual`: numerical vector of actual values

**Value**

A numeric vector of performance measures.

**Examples**

```r
data(titanic)  
titanic <- titanic[complete.cases(titanic),]  
model <- partykit::ctree(Survived ~ Sex + Pclass, data = titanic)  
pred <- predict(model, type = "prob")[, "Yes"]  
PerfsBinClassif(pred, titanic$Survived)
```

PerfsRegression  
*Performance measures for regressions*

**Description**

Computes various performance measures for regression tasks: sum of the squared errors (SSE), mean squared errors (MSE), root mean squared errors (RMSE), coefficient of determination (R2), Kendall’s rank correlation (tau).

**Usage**

```r
PerfsRegression(pred, actual)
```

**Arguments**

- `pred`: numerical vector of predicted values
- `actual`: numerical vector of actual values
Prototypes

Value
A numeric vector of performance measures.

Examples

```r
data(titanic)
titanic <- titanic[complete.cases(titanic),]
model <- partykit::ctree(Age ~ Sex + Pclass, data = titanic)
pred <- predict(model)
PerfsRegression(pred, titanic$Age)
```

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

```r
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 neighbors 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#prototype

Examples
```r
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
```r
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 with the 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


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

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

---

**TreeStab**

*Stability assessment of conditional inference trees*

**Description**
Assesses the stability of conditional inference trees through the partition of observations in the terminal nodes and the frequency of the variables used for splits.

**Usage**

```r
TreeStab(ct, B = 20)
```

**Arguments**

- **ct**  A tree of class `constparty` (as returned by `ctree` from `partykit` package).
- **B**  Numerical value. The number of bootstrap replications. Default is 20.

**Details**
The study of splitting variables used in the original tree and in bootstrap trees in directly inspired from the approach implemented in `stablelearner` package. The other side of this functions also uses bootstrap trees, this time to compute the Jaccard index of concordance between partitions, to assess the stability of the partition of observations in the terminal nodes of the tree.

**Value**
A list of two elements:

- **partition**  average Jaccard index of concordance between the partition (terminal nodes) of `ct` and the partitions of bootstrap trees
- **variables**  a data frame with splitting variables in rows and two statistics in columns: their frequency of use in the tree vs in the bootstrap trees, and
TreeStab

Author(s)
Nicolas Robette

References

See Also
ctree

Examples

data(iris)
iris2 = iris
iris2$Species = factor(iris$Species == "versicolor")
iris.ct = partykit::ctree(Species ~ ., data = iris2)
TreeStab(iris.ct, B = 10)
Index

* **aplot**
  ggForestEffects, 19
  ggVarImp, 20
* **classif**
  Outliers, 25
  Prototypes, 27
* **datasets**
  titanic, 29
* **tree**
  ctree-module, 4
  EasyTreeVarImp, 5
  fastcforest, 6
  GetAleData, 13
  GetCtree, 14
  GetInteractionStrength, 15
  GetPartialData, 16
  GetSplitStats, 18
  ggForestEffects, 19
  ggVarImp, 20
  ictree, 21
  NiceTreePlot, 22
  NodesInfo, 23
  NodeTreePlot, 24
  PerfsBinClassif, 26
  PerfsRegression, 26
  TreeStab, 30

BivariateAssoc, 2

cforest, 7, 9, 13, 15, 16
cforest_control, 6
ctree-module, 4
ctreeServer (ctree-module), 4
ctreeUI (ctree-module), 4

EasyTreeVarImp, 5

fastcforest, 6, 9, 13, 15, 16
fastvarImp, 7, 8, 21
fastvarImpAUC, 9, 9, 21

FeatureEffect, 13
FeatureSelection, 11
foreach, 13, 17
GetAleData, 13, 15, 17, 20
GetCtree, 14
GetInteractionStrength, 13, 15, 17
GetPartialData, 13, 15, 16, 20
GetSplitStats, 18
ggForestEffects, 19
ggVarImp, 20

ictree, 4, 21
iml, 13

NiceTreePlot, 22
NodesInfo, 23
NodeTreePlot, 24

Outliers, 25

partial, 17
pdp, 17

PerfsBinClassif, 26
PerfsRegression, 26
Prototypes, 27

ptrafo, 7

shiny::callModule(), 4
shiny::reactive(), 4

SurrogateTree, 28

titanic, 29
TreeStab, 30

varImp, 9, 21
varImpAUC, 21
vip, 15