Package 'etree'

July 16, 2022

Title Classification and Regression with Structured and Mixed-Type Data

Version 0.1.0

Description Implementation of Energy Trees, a statistical model to perform classification and regression with structured and mixed-type data. The model has a similar structure to Conditional Trees, but brings in Energy Statistics to test independence between variables that are possibly structured and of different nature. Currently, the package covers functions and graphs as structured covariates. It builds upon 'partykit' to provide functionalities for fitting, printing, plotting, and predicting with Energy Trees. Energy Trees are described in Giubilei et al. (2022) <arXiv:2207.04430>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Depends R (>= 3.7.0)

Imports brainGraph, cluster, energy, fda.usc (>= 2.0.0), igraph, NetworkDistance, parallel, partykit, survival, TDA, usedist

Suggests knitr, MLmetrics, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/ricgbl/etree

BugReports https://github.com/ricgbl/etree/issues

NeedsCompilation no

Author Riccardo Giubilei [aut, cre] (<https://orcid.org/0000-0002-1674-4886>),
Tullia Padellini [aut],
Pierpaolo Brutti [aut],
Marco Brandi [ctb],
Gabriel Nespoli [ctb],
Torsten Hothorn [ctb] (<https://orcid.org/0000-0001-8301-0471>,

(partykit author)),
Achim Zeileis [ctb] (<https://orcid.org/0000-0003-0918-3766>, (partykit author))

Maintainer Riccardo Giubilei <riccardogbl@gmail.com>

Repository CRAN

Date/Publication 2022-07-16 08:30:02 UTC

R topics documented:

`etree-package` *etree: Classification and Regression With Structured and Mixed-Type Data*

Description

Implementation of Energy Trees, a statistical model to perform classification and regression with structured and mixed-type data. The model has a similar structure to Conditional Trees, but brings in Energy Statistics to test independence between variables that are possibly structured and of different nature. Currently, the package covers functions and graphs as structured covariates. It builds upon 'partykit' to provide functionalities for fitting, printing, plotting, and predicting with Energy Trees.

data_cls *Classification toy dataset*

Description

A simple dataset containing simulated values for a nominal response variable and four covariates of both mixed and partially structured type. The data generation process is based on Example 4.7 (”Signal shape classification”, pages 73-77) from Saito (1994).

Usage

```
data_cls
```

Format

List with two elements: `covs`, which is a list containing the covariates, and `resp`, which is a factor of length 150 representing the response variable. The response variable is divided into three classes whose labels are cylinder (`Cyl`), bell (`Bel`) and funnel (`Fun`). The four covariates in `covs` all have length 150 and are characterized as follows:

- Nominal: `Cyl` observations are given level 1 with probability 0.8 and levels 2 and 3 with probability 0.1 each, `Bel` observations are given level 2 with probability 0.8 and levels 1 and 3 with probability 0.1 each, `Fun` observations are given level 3 with probability 0.8 and levels 1 and 2 with probability 0.1 each;
- Numeric: coefficients for one of the basis used to perform the B-splines expansion of the curves that are in turn specified as in Saito (1994);
- Functional: curves as specified in Saito (1994);
- Graphs: Erd\"os-R\'enyi graphs with connection probability 0.10 for `Cyl` observations, 0.125 for `Bel` observations, 0.15 for `Fun` observations.

References

Saito, N. (1994). Local feature extraction and its applications using a library of bases (Doctoral dissertation, Yale University).

data_reg *Regression toy dataset*

Description

A simple dataset containing simulated values for a numeric response variable and four covariates of both mixed and partially structured type. The data generation process is based on Section 5 (”Example: synthetic data”) from Serban and Wasserman (2005).

Usage

```
data_reg
```

Format

List with two elements: `covs`, which is a list containing the covariates, and `resp`, which is a numeric vector of length 200 representing the response variable. The response variable is specified as in Serban and Wasserman (2005). The four covariates in `covs` all have length 200 and are characterized as follows:

- Nominal: level 0 for observations having negative response variable, level 1 otherwise;
- Numeric: coefficients for one of the basis used to perform the B-splines expansion of the curves that are in turn specified as in Serban and Wasserman (2005);
- Functional: curves as specified in Serban and Wasserman (2005), with 50 observations coming from each of the four curve shapes;
- Graphs: Erd\"os-R\'enyi graphs with connection probability given by a transformation of the response variable obtained standardizing between 0.2 and 0.8 its value after adding a normally distributed noise with mean 0 and standard deviation 7.

References

Serban, N., and Wasserman, L. (2005). CATS: clustering after transformation and smoothing. *Journal of the American Statistical Association*, 100(471), 990-999.

`dist_comp` *Distances*

Description

Compute pairwise distances starting from single objects containing the original univariate observations.

Usage

```
dist_comp(x, lp = 2)
```

Arguments

`x`	Object containing the original univariate observations. Currently available types and the form they need to have to be correctly recognized are the following: • Logical: logical vectors; • Numeric: numeric or integer vectors; • Nominal: factors; • Functions: objects of class `"fdata"`; • Graphs: (lists of) objects of class `"igraph"`;

- Persistence diagrams: (lists of) objects with `attributes(x)$names == "diagram"`.

See Details to find out which distance is used in each case.

`lp` Integer specifying which norm should be used to compute the distances for functional data.

Details

The distances used in each case are the following:

- Logical: Euclidean distance, implemented via `dist()`;
- Numeric: Euclidean distance, implemented via `dist()`;
- Nominal: Gower's distance, implemented via `daisy()`;
- Functions: L^p-norm, implemented via `metric.lp()` with default options;
- Graphs: Edge Difference distance (Hammond et al., 2013), implemented via `nd.edd()`;
- Persistence diagrams: Wasserstein distance, implemented via `wasserstein()` with default options;

Value

Object of class `"dist"` containing the pairwise distances.

References

D. K. Hammond, Y. Gur, and C. R. Johnson (2013). Graph diffusion distance: A difference measure for weighted graphs based on the graph laplacian exponential kernel. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 419-422.

Examples

```
# Number of observations
nobs <- 10

## Logical
obj <- as.logical(rbinom(nobs, 1, 0.5))
d <- dist_comp(obj)

## Integer
obj <- rpois(nobs, 5)
d <- dist_comp(obj)

## Numeric
obj <- rnorm(nobs)
d <- dist_comp(obj)

## Factors
obj <- factor(letters[1:nobs])
d <- dist_comp(obj)
```

```
## Functional data
obj <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
d <- dist_comp(obj)

## Graphs
obj <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
d <- dist_comp(obj)

## Persistence diagrams
x <- lapply(rep(100, nobs), function(np) TDA::circleUnif(np))
obj <- lapply(x, TDA::ripsDiag, maxdimension = 1, maxscale = 3)
d <- dist_comp(obj)
```

`eforest` *Energy Forests*

Description

Fits an Energy Forest, in the form of either a bagging of Energy Trees or a Random Energy Forest, depending on the value of the `random_covs` parameter.

Usage

```
eforest(
  response,
  covariates,
  weights = NULL,
  ntrees = 100,
  ncores = 1L,
  minbucket = 1,
  alpha = 1,
  R = 500,
  split_type = "cluster",
  coeff_split_type = "test",
  p_adjust_method = "fdr",
  perf_metric = NULL,
  random_covs = "auto",
  verbose = FALSE
)
```

Arguments

`response`	Response variable, an object of class either `"factor"` or `"numeric"` (for classification and regression, respectively).
`covariates`	Set of covariates. Must be provided as a list, where each element is a different variable. Currently available types and the form they need to have to be correctly recognized are the following:

- Numeric: numeric or integer vectors;
- Nominal: factors;
- Functions: objects of class `"fdata"`;
- Graphs: (lists of) objects of class `"igraph"`.

Each element (i.e., variable) in the covariates list must have the same `length()`, which corresponds to the sample size.

`weights` Optional vector of non-negative integer-valued weights to be used in the fitting process. If not provided, all observations are assumed to have weight equal to 1.

`ntrees` Number of Energy Trees to grow, i.e., the number of bootstrap samples to be generated and used for fitting.

`ncores` Number of cores to use, i.e., at most how many child processes will be run simultaneously. Must be exactly 1 on Windows (which uses the master process). `ncores` corresponds to `mc.cores` in `mclapply()`, which is actually used to grow the single Energy Trees in a parallel fashion.

`minbucket` Positive integer specifying the minimum number of observations that each terminal node must contain. Default is 5.

`alpha` Nominal level controlling the probability of type I error in the Energy tests of independence used for variable selection. Default is 0.05.

`R` Number of replicates employed to approximate the sampling distribution of the test statistic in every Energy test of independence. Default is 1000.

`split_type` Splitting method used when the selected covariate is structured. It has two possible values: `"coeff"` for feature vector extraction, and `"cluster"` for clustering. See Details for further information.

`coeff_split_type`
Method to select the split point for the chosen component when the selected covariate is structured and `split_type = "coeff"`. It has two possible values: `"test"`, in which case Energy tests of independence are used, and `"traditional"`, to employ traditional methods (Gini index for classification and RSS for regression). See Details for further information.

`p_adjust_method`
Multiple-testing adjustment method for P-values, which can be set to any of the values provided by `p.adjust.methods`. Default is `"fdr"` for False Discovery Rate.

`perf_metric` Performance metric that is used to compute the Out-Of-Bag score. If `NULL`, default choices are used: Balanced Accuracy for classification and Root Mean Square Percentage Error for regression. See Details for further information and possible alternatives.

`random_covs` Size of the random subset of covariates to choose from at each split. If set to `NULL`, all the covariates are considered each time, resulting in a bagging of Energy Trees. When `random_covs` is an integer greater than 1 and less than the total number of covariates, the model is a Random Energy Forest. By default, it is equal to `"auto"`, which implies the square root of the number of covariates for classification, or one third of the number of covariates for regression (in both cases, rounded down to the nearest integer).

`verbose` Logical indicating whether to print a one-line notification for the conclusion of each tree's fitting process.

Details

`eforest()` generates `ntrees` bootstrap samples and then calls `etree()` on each of them. Then, it computes the Out-Of-Bag (OOB) score using the performance metric defined through `perf_metric`.

For classification, possible values of `perf_metric` are `"BAcc"` and `"WBAcc"`. Both are general enough to be used in multiclass classification problems, still producing sensible results in the case of binary classification. The two options are based on the calculation of a ground performance metric, the Balanced Accuracy, which is defined as the arithmetic mean between Sensitivity and Specificity. In this framework, Balanced Accuracy is computed using a "One vs. All" approach, i.e., considering one class at a time: positive instances are those belonging to that class, and negatives are the ones belonging to any other class. Then, the "One vs. All" Balanced Accuracies obtained by considering each class must be averaged. When `perf_metric = "BAcc"` (default for classification tasks), the average is arithmetic. When `perf_metric = "WBAcc"`, the average is weighted using class sizes, hence giving more importance to the "One vs. All" Balanced Accuracy of larger classes.

For regression, the default value of `perf_metric` is `"RMSPE"`, namely, Root Mean Square Percentage Error. Other available options are `c("MAE", "MAPE", "MedianAE", "MedianAPE", "MSE", "NRMSE", "RAE","RMSE", "RMLSE")`. Each of these name points to the corresponding homonym function from the package `MLmetrics`, whose documentation provides more information about their definition.

Value

Object of class `"eforest"` with three elements: 1) `ensemble`, which is a list gathering all the fitted trees; 2) `oob_score`, an object of class `"numeric"` representing the OOB score computed using the performance metric defined through `perf_metric`; 3) `perf_metric`, an object of class `"character"` returning the performance metric used for computations.

Examples

```
## Covariates
set.seed(123)
nobs <- 100
cov_num <- rnorm(nobs)
cov_nom <- factor(rbinom(nobs, size = 1, prob = 0.5))
cov_gph <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
cov_fun <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
cov_list <- list(cov_num, cov_nom, cov_gph, cov_fun)

## Response variable(s)
resp_reg <- cov_num ^ 2
y <- round((cov_num - min(cov_num)) / (max(cov_num) - min(cov_num)), 0)
resp_cls <- factor(y)

## Regression ##
eforest_fit <- eforest(response = resp_reg, covariates = cov_list, ntrees = 12)
print(eforest_fit$ensemble[[1]])
plot(eforest_fit$ensemble[[1]])
mean((resp_reg - predict(eforest_fit)) ^ 2)
```

```
## Classification ##
eforest_fit <- eforest(response = resp_cls, covariates = cov_list, ntrees = 12)
print(eforest_fit$ensemble[[12]])
plot(eforest_fit$ensemble[[12]])
table(resp_cls, predict(eforest_fit))
```

`etree` *Energy Tree*

Description

Fits an Energy Tree for classification or regression.

Usage

```
etree(
  response,
  covariates,
  weights = NULL,
  minbucket = 5,
  alpha = 0.05,
  R = 1000,
  split_type = "coeff",
  coeff_split_type = "test",
  p_adjust_method = "fdr",
  random_covs = NULL
)
```

Arguments

`response`	Response variable, an object of class either `"factor"` or `"numeric"` (for classification and regression, respectively).
`covariates`	Set of covariates. Must be provided as a list, where each element is a different variable. Currently available types and the form they need to have to be correctly recognized are the following: • Numeric: numeric or integer vectors; • Nominal: factors; • Functions: objects of class `"fdata"`; • Graphs: (lists of) objects of class `"igraph"`. Each element (i.e., variable) in the covariates list must have the same `length()`, which corresponds to the sample size.
`weights`	Optional vector of non-negative integer-valued weights to be used in the fitting process. If not provided, all observations are assumed to have weight equal to 1.

`minbucket` Positive integer specifying the minimum number of observations that each terminal node must contain. Default is 5.

`alpha` Nominal level controlling the probability of type I error in the Energy tests of independence used for variable selection. Default is 0.05.

`R` Number of replicates employed to approximate the sampling distribution of the test statistic in every Energy test of independence. Default is 1000.

`split_type` Splitting method used when the selected covariate is structured. It has two possible values: `"coeff"` for feature vector extraction, and `"cluster"` for clustering. See Details for further information.

`coeff_split_type` Method to select the split point for the chosen component when the selected covariate is structured and `split_type = "coeff"`. It has two possible values: `"test"`, in which case Energy tests of independence are used, and `"traditional"`, to employ traditional methods (Gini index for classification and RSS for regression). See Details for further information.

`p_adjust_method` Multiple-testing adjustment method for P-values, which can be set to any of the values provided by `p.adjust.methods`. Default is `"fdr"` for False Discovery Rate.

`random_covs` Size of the random subset of covariates to choose from at each split. If set to `NULL` (default), all the covariates are considered each time.

Details

`etree()` is the main function of the homonym package. It allows implementing Energy Trees by simply specifying the response variable, the set of covariates, and possibly some other parameters. The function is specified in the same way regardless of the task type: the choice between classification and regression is automatically made depending on the nature of the response variable.

Energy Trees (Giubilei et al., 2022) are a recursive partitioning tree-based model built upon Conditional Trees (Hothorn et al., 2006). At each step of Energy Trees' iterative procedure, an Energy test of independence (Szekely et al., 2007) is performed between the response variable and each of the J covariates. If the test of global independence (defined as the intersection of the J tests of partial independence) is not rejected at the significance level set by `alpha`, the recursion is stopped; otherwise, the covariate most associated with the response in terms of P-value is selected for splitting. When the covariate is traditional (i.e, numeric or nominal), an Energy test of independence is performed for each possible split point, and the one yielding the strongest association with the response is chosen. When the selected covariate is structured, the split procedure is defined by the value of `split_type`, and possibly by that of `coeff_split_type`.

`split_type` specifies the splitting method for structured covariates. It has two possible values:

- `"coeff"`: in this case, feature vector extraction is used to transform the structured selected covariate into a set of numeric components using a representation that is specific to its type. Available transformations of such a kind are cubic B-spline expansions for functional data and shell distributions (Carmi et al., 2007) for graphs - obtained through k-cores (Seidman, 1983), s-cores (Eidsaa and Almaas, 2013), and d-cores (Giatsidis et al., 2013), for binary, weighted, and directed graphs, respectively. Then, the component most associated with the response is selected using Energy tests of independence (Szekely et al., 2007), and the split point for that component is chosen using the method defined by `coeff_split_type`;

- "cluster": in this case, the observed values for the structured selected covariate are used within a Partitioning Around Medoids (Kaufmann and Rousseeuw, 1987) step to split observations into the two kid nodes. Medoids calculation and units assignment are performed using pam(). Distances are specific to each type of variable (see dist_comp() for details).

coeff_split_type defines the method to select the split point for the chosen component of the selected structured covariate if and only if split_type = "coeff". It has two possible values:

- "test": an Energy test of independence (Szekely et al., 2007) is performed for each possible split point of the chosen component, and the one yielding the strongest association with the response is selected;
- "traditional": the split point for the chosen component is selected as the one minimizing the Gini index (for classification) or the RSS (for regression) in the two kid nodes.

Value

An object of class "etree", "constparty", and "party". It stores all the information about the fitted tree. Its elements can be individually accessed using the $ operator. Their names and content are the following:

- node: a partynode object representing the basic structure of the tree;
- data: a list containing the data used for the fitting process. Traditional covariates are included in their original form, while structured covariates are stored in the form of components if split_type = "coeff" or as a factor whose levels go from 1 to the total number of observations if split_type = "cluster";
- fitted: a data.frame whose number of rows coincides with the sample size. It includes the fitted terminal node identifiers (in "(fitted)") and the response values of all observations (in "(response)");
- terms: a terms object;
- names (optional): names of the nodes in the tree. They can be set using a character vector: if its length is smaller than the number of nodes, the remaining nodes have missing names; if its length is larger, exceeding names are ignored.

References

R. Giubilei, T. Padellini, P. Brutti (2022). Energy Trees: Regression and Classification With Structured and Mixed-Type Covariates. arXiv preprint. https://arxiv.org/pdf/2207.04430.pdf.

S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir (2007). A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences*, 104(27):11150-11154.

M. Eidsaa and E. Almaas (2013). S-core network decomposition: A generalization of k-core analysis to weighted networks. *Physical Review E*, 88(6):062819.

C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis (2013). D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems*, 35(2):311-343.

T. Hothorn, K. Hornik, and A. Zeileis (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651-674.

L. Kaufmann and P. Rousseeuw (1987). Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405-416.

S. B. Seidman (1983). Network structure and minimum degree. *Social networks*, 5(3):269-287.

G. J. Szekely, M. L. Rizzo, and N. K. Bakirov (2007). Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769-2794.

See Also

`ctree()` for the `partykit` implementation of Conditional Trees (Hothorn et al., 2006).

Examples

```
## Covariates
nobs <- 100
cov_num <- rnorm(nobs)
cov_nom <- factor(rbinom(nobs, size = 1, prob = 0.5))
cov_gph <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
cov_fun <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
cov_list <- list(cov_num, cov_nom, cov_gph, cov_fun)

## Response variable(s)
resp_reg <- cov_num ^ 2
y <- round((cov_num - min(cov_num)) / (max(cov_num) - min(cov_num)), 0)
resp_cls <- factor(y)

## Regression ##
etree_fit <- etree(response = resp_reg, covariates = cov_list)
print(etree_fit)
plot(etree_fit)
mean((resp_reg - predict(etree_fit)) ^ 2)

## Classification ##
etree_fit <- etree(response = resp_cls, covariates = cov_list)
print(etree_fit)
plot(etree_fit)
table(resp_cls, predict(etree_fit))
```

`etree-methods` *Methods for "etree" objects*

Description

Methods for objects of class `"etree"`.

Usage

```
## S3 method for class 'etree'
print(
  x,
  FUN = NULL,
  digits = getOption("digits") - 4,
  header = NULL,
  footer = TRUE,
  ...
)

## S3 method for class 'etree'
length(x)

## S3 method for class 'etree'
depth(x, root = FALSE, ...)

## S3 method for class 'etree'
width(x, ...)

## S3 method for class 'etree'
x[i, ...]

## S3 method for class 'etree'
x[[i, ...]]
```

Arguments

x	Object of class "etree".
FUN	Function to be applied to nodes.
digits	Number of digits to be printed.
header	Header to be printed.
footer	Footer to be printed.
...	Additional arguments.
root	Logical indicating whether the root node should be counted in depth() or not (default).
i	Integer specifying the root of the subtree to extract.

Value

The print() method generates a textual representation of the tree. length() returns the number of nodes in the tree, depth() the depth of the tree and width() the number of terminal nodes. The subset methods extract subtrees starting from a given node.

Functions

- `print.etree`: Generates textual representation of the tree.
- `length.party`: Number of nodes in the tree.
- `depth.party`: Depth of the three.
- `width.party`: Number of terminal nodes.
- `[.etree`: Extract subtrees.
- `[[.etree`: Extract subtrees.

Examples

```
## Covariates
nobs <- 100
cov_num <- rnorm(nobs)
cov_nom <- factor(rbinom(nobs, size = 1, prob = 0.5))
cov_gph <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
cov_fun <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
cov_list <- list(cov_num, cov_nom, cov_gph, cov_fun)

## Response variable
resp_reg <- cov_num ^ 2

## Fit
etree_fit <- etree(response = resp_reg, covariates = cov_list)

## Print
print(etree_fit)

## Number of nodes in the tree
length(etree_fit)

## Depth of the tree
depth(etree_fit)

## Number of terminal nodes in the tree
width(etree_fit)

## Extract subtrees
etree_fit[2]
etree_fit[[2]]
```

etree-size *Size of Energy Trees*

Description

Depth and width of an Energy Tree.

Usage

```
depth(x, ...)

width(x, ...)
```

Arguments

x	An object of class etree.
...	Additional arguments.

Value

depth() returns the depth of the tree and width() gives the number of terminal nodes.

Functions

- depth: Depth of the three.
- width: Number of terminal nodes in the tree.

nodeapply *Apply functions over nodes*

Description

Returns a list of values obtained by applying a function to "etree" or "partynode" objects.

Usage

```
nodeapply(obj, ids = 1, FUN = NULL, ...)

## S3 method for class 'partynode'
nodeapply(obj, ids = 1, FUN = NULL, ...)

## S3 method for class 'etree'
nodeapply(obj, ids = 1, FUN = NULL, by_node = TRUE, ...)
```

Arguments

obj	Object of class "etree" or "partynode".
ids	Integer vector of node identifiers to apply over.
FUN	Function to be applied to nodes. By default, the node itself is returned.
...	Additional arguments.
by_node	Logical indicating whether FUN should be applied to subsets of "partynode" objects (default) or not, in which case it is applied to subsets of "etree" objects.

Details

The method for "partynode" objects apply function FUN to all nodes with node identifiers in ids. The method for "etree" objects by default calls the nodeapply method on the corresponding node slot. If by_node is FALSE, it is applied to the "etree" object with root node ids.

Value

A list of results whose length is given by length(ids).

Methods (by class)

- partynode: nodeapply() method for objects of class "partynode".
- etree: nodeapply() method for objects of class "etree".

Examples

```
## Covariates
nobs <- 100
cov_num <- rnorm(nobs)
cov_nom <- factor(rbinom(nobs, size = 1, prob = 0.5))
cov_gph <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
cov_fun <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
cov_list <- list(cov_num, cov_nom, cov_gph, cov_fun)

## Response variable
resp_reg <- cov_num ^ 2

## Fit
etree_fit <- etree(response = resp_reg, covariates = cov_list)

## Get pvalues of inner nodes
tnodes <- nodeids(etree_fit, terminal = TRUE)
nodes <- 1:max(tnodes)
inodes <- nodes[-tnodes]
nodeapply(etree_fit, ids = inodes, FUN = function(n) n$info$pvalue)
```

nodeids *Extract node identifiers.*

Description

Extract unique identifiers from inner and terminals nodes of "etree" or "partynode" objects.

Usage

```
nodeids(obj, ...)

## S3 method for class 'partynode'
nodeids(obj, from = NULL, terminal = FALSE, ...)

## S3 method for class 'etree'
nodeids(obj, from = NULL, terminal = FALSE, ...)
```

Arguments

obj	Object of class "etree" or "partynode".
...	Additional arguments.
from	Integer specifying the node to start from.
terminal	Logical indicating whether only identifiers of terminal nodes should be returned (FALSE by default).

Value

An integer vector of node identifiers.

Methods (by class)

- partynode: nodeids() method for objects of class "partynode".
- etree: nodeids() method for objects of class "etree".

Examples

```
## Covariates
nobs <- 100
cov_num <- rnorm(nobs)
cov_nom <- factor(rbinom(nobs, size = 1, prob = 0.5))
cov_gph <- lapply(1:nobs, function(j) igraph::sample_gnp(100, 0.2))
cov_fun <- fda.usc::rproc2fdata(nobs, seq(0, 1, len = 100), sigma = 1)
cov_list <- list(cov_num, cov_nom, cov_gph, cov_fun)
```

```
## Response variable
resp_reg <- cov_num ^ 2

## Fit
etree_fit <- etree(response = resp_reg, covariates = cov_list)

## Get all nodes identifiers
nodes_ids <- nodeids(etree_fit)

## Get terminal nodes identifiers
tnodes_ids <- nodeids(etree_fit, terminal = TRUE)

## Get all nodes identifiers starting from 2
nodes_ids2 <- nodeids(etree_fit, from = 2)
```

`plot.etree` *Visualization of Energy Trees*

Description

Returns the plot of an object of class `"etree"`.

Usage

```
## S3 method for class 'etree'
plot(
  x,
  main = NULL,
  terminal_panel = NULL,
  tp_args = list(),
  inner_panel = node_inner,
  ip_args = list(),
  edge_panel = edge_simple,
  ep_args = list(),
  type = c("extended", "simple"),
  drop_terminal = NULL,
  tnex = NULL,
  newpage = TRUE,
  pop = TRUE,
  gp = gpar(),
  ...
)
```

Arguments

`x`	An object of class `"etree"`, i.e., a fitted Energy Tree.
`main`	Optional title for the plot.

terminal_panel	Optional panel function of the form function(node) plotting the terminal nodes. Alternatively, a panel generating function of class "grapcon_generator" that is called with arguments x and tp_args to set up a panel function. By default, an appropriate panel function is chosen depending on the scale of the dependent variable.
tp_args	List of arguments passed to terminal_panel if this is a "grapcon_generator" object.
inner_panel	Optional panel function of the form function(node) plotting the inner nodes. Alternatively, a panel generating function of class "grapcon_generator" that is called with arguments x and ip_args to set up a panel function.
ip_args	List of arguments passed to inner_panel if this is a "grapcon_generator" object.
edge_panel	Optional panel function of the form function(split, ordered = FALSE, left = TRUE) plotting the edges. Alternatively, a panel generating function of class "grapcon_generator" that is called with arguments x and ep_args to set up a panel function.
ep_args	List of arguments passed to edge_panel if this is a "grapcon_generator" object.
type	Character specifying the complexity of the plot: extended tries to visualize the distribution of the response variable in each terminal node whereas simple only gives some summary information.
drop_terminal	Logical indicating whether all terminal nodes should be plotted at the bottom.
tnex	Numeric value giving the terminal node extension in relation to the inner nodes.
newpage	Logical indicating whether grid.newpage() should be called.
pop	Logical indicating whether the viewport tree should be popped before return.
gp	Graphical parameters.
...	Additional arguments.

Details

The plot() method for "etree" objects allows for the visualization of fitted Energy Trees, as returned by etree() or as contained in the ensemble element of a fitted Random Energy Forest.

Value

No return value, called for side effects (plotting the tree).

predict.eforest *Predictions for Energy Forests*

Description

Compute predictions for objects of class "eforest" (i.e., as returned by eforest()).

Usage

```
## S3 method for class 'eforest'
predict(object, newdata = NULL, ...)
```

Arguments

object	A fitted Energy Forest of class "eforest".
newdata	Optional set of new covariates used to make predictions. Must be provided as a list, where each element is a different variable. Currently available types and the form they need to have to be correctly recognized are the following: • Numeric: numeric or integer vectors; • Nominal: factors; • Functions: objects of class "fdata"; • Graphs: (lists of) objects of class "igraph". Each element (i.e., variable) in the covariates list must have the same length(), which corresponds to the (new) sample size. If newdata is omitted, fitted values of individual trees are somehow combined (see Details) and returned.
...	Additional arguments.

Details

The predict() method for "eforest" objects computes predictions for Energy Forests as returned by eforest(). Predictions are based either on the fitted values (if newdata is NULL) or on the new set of covariates (when newdata is provided). In both cases, each tree in object$ensemble is used to make predictions by calling predict() on it (with the same specification of newdata). Then, individual trees' predictions for any single observation are combined by majority voting rule for classification or by arithmetic mean for regression.

Value

Predictions, in the form of a factor for classification or as a numeric vector for regression.

predict.etree *Predictions for Energy Trees*

Description

Compute predictions for objects of class "etree" (i.e., fitted Energy Trees as returned by etree(), or as contained in the ensemble element of a fitted Random Energy Forest).

Usage

```
## S3 method for class 'etree'
predict(object, newdata = NULL, perm = NULL, ...)
```

Arguments

object	A fitted Energy Tree of class "etree".
newdata	Optional set of new covariates used to make predictions. Must be provided as a list, where each element is a different variable. Currently available types and the form they need to have to be correctly recognized are the following: • Numeric: numeric or integer vectors; • Nominal: factors; • Functions: objects of class "fdata"; • Graphs: (lists of) objects of class "igraph". Each element (i.e., variable) in the covariates list must have the same length(), which corresponds to the (new) sample size. If newdata is omitted, fitted values are returned.
perm	Optional character vector of variable names. Splits of nodes with a primary split in any of these variables will be permuted (after dealing with surrogates). Note that surrogate split in the perm variables will not be permuted.
...	Additional arguments.

Details

The predict() method for "etree" objects yields predictions for fitted Energy Trees as returned by etree() or as contained in the ensemble element of a fitted Random Energy Forest. Predictions are based either on fitted values (if newdata is NULL) or on the new set of covariates (if newdata is provided). The values of split_type and coeff_split_type, as well as the number of components for each structured covariate (needed to compute an equivalent representation for the covariates in newdata when split_type = "coeff"), are automatically retrieved from the object of class "etree".

Value

Predictions, in the form of a factor for classification or as a numeric vector for regression.

Index