Package: ODRF (via r-universe)

October 9, 2024

Type Package

Title Oblique Decision Random Forest for Classification and Regression

Version 0.0.4.9002

Author Yu Liu [aut, cre, cph], Yingcun Xia [aut]

Maintainer Yu Liu <liuyuchina123@gmail.com>

Description The oblique decision tree (ODT) uses linear combinations of predictors as partitioning variables in a decision tree. Oblique Decision Random Forest (ODRF) is an ensemble of multiple ODTs generated by feature bagging. Oblique Decision Boosting Tree (ODBT) applies feature bagging during the training process of ODT-based boosting trees to ensemble multiple boosting trees. All three methods can be used for classification and regression, and ODT and ODRF serve as supplements to the classical CART of Breiman (1984) <DOI:10.1201/9781315139470> and Random Forest of Breiman (2001) <DOI:10.1023/A:1010933404324> respectively.

License GPL (>= 3)

URL https://liuyu-star.github.io/ODRF/

BugReports https://github.com/liuyu-star/ODRF/issues

Depends partykit, R (>= 3.5.0)

Imports doParallel, foreach, glue, graphics, grid, lifecycle, magrittr, nnet, parallel, Pursuit, Rcpp, rlang (>= 0.4.11), stats, rpart, glmnet

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

LinkingTo Rcpp, RcppArmadillo, RcppEigen

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData yes

NeedsCompilation yes

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://liuyu-star.r-universe.dev

RemoteUrl https://github.com/liuyu-star/odrf

RemoteRef HEAD

RemoteSha 863dfa0b0b8f58ec7f3fe503e0b0561ed030218a

Contents

Accuracy *accuracy of oblique decision random forest*

Description

Prediction accuracy of ODRF at different tree sizes.

Usage

```
Accuracy(obj, data, newdata = NULL)
```

Arguments

obj	An object of class ODRF, as that created by the function ODRF.
data	Training data of class data.frame in ODRF is used to calculate the OOB error.
newdata	A data frame or matrix containing new data is used to calculate the test error. If it is missing, then it is replaced by data.

Value

OOB error and test error, misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

ODRF VarImp plot.Accuracy

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 80)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, split = "gini",
parallel = FALSE, ntrees = 50)
(error <- Accuracy(forest, train_data, test_data))
```

as.party.ODT ODT *as* party

Description

To make ODT object to objects of class party.

Usage

```
## S3 method for class 'ODT'
as.party(obj, data, ...)
```

Arguments

obj	An object of class ODT.
data	Training data of class data.frame is used to convert the object of class ODT, and it must be the training data data in ODT.
...	Arguments to be passed to methods

Value

An objects of class party.

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software.

See Also

ODT party

Examples

```
data(iris)
tree <- ODT(Species ~ ., data = iris)
tree
plot(tree)
party.tree <- as.party(tree, data = iris)
party.tree
plot(party.tree)
```

`best.cut.node` *find best splitting variable and node*

Description

A function to select the splitting variables and nodes using one of four criteria.

Usage

```
best.cut.node(
  X,
  y,
  Xsplit = X,
  split,
  lambda = "log",
  weights = 1,
  MinLeaf = 10,
  numLabels = ifelse(split %in% c("gini", "entropy"), length(unique(y)), 0),
  glmnetParList = NULL
)
```

Arguments

`X`	An n by d numeric matrix (preferable) or data frame.
`y`	A response vector of length n.
`Xsplit`	Splitting variables used to construct linear model trees. The default value is NULL and is only valid when split="linear".
`split`	The criterion used for splitting the nodes. "entropy": information gain and "gini": gini impurity index for classification; "": mean square error for regression; "linear": mean square error for multiple linear regression.
`lambda`	The argument of `split` is used to determine the penalty level of the partition criterion. Three options are provided including, `lambda=0`: no penalty; `lambda=2`: AIC penalty; `lambda='log'` (Default): BIC penalty. In Addition, lambda can be any value from 0 to n (training set size).
`weights`	A vector of values which weigh the samples when considering a split.
`MinLeaf`	Minimal node size (Default 10).
`numLabels`	The number of categories.
`glmnetParList`	List of parameters used by the functions `glmnet` and `cv.glmnet` in package `glmnet`. If left unchanged, default values will be used, for details see `glmnet` and `cv.glmnet`.

Value

A list which contains:

- BestCutVar: The best split variable.
- BestCutVal: The best split points for the best split variable.
- BestIndex: Each variable corresponds to maximum decrease in gini impurity index, information gain, and mean square error.
- fitL and fitR: The multivariate linear models for the left and right nodes after splitting are trained using the function glmnet.

Examples

```
### Find the best split variable ###
#Classification
data(iris)
X <- as.matrix(iris[, 1:4])
y <- iris[[5]]
(bestcut <- best.cut.node(X, y, split = "gini"))
(bestcut <- best.cut.node(X, y, split = "entropy"))

#Regression
data(body_fat)
X=body_fat[, -1]
y=body_fat[, 1]
(bestcut <- best.cut.node(X, y, split = "mse"))

set.seed(10)
cutpoint=50
X=matrix(rnorm(100*10),100,10)
age=sample(seq(20,80),100,replace = TRUE)
height=sample(seq(50,200),100,replace = TRUE)
weight=sample(seq(5,150),100,replace = TRUE)
Xsplit=cbind(age=age,height=height,weight=weight)
mu=rep(0,100)
mu[age<=cutpoint]=X[age<=cutpoint,1]+X[age<=cutpoint,2]
mu[age>cutpoint]=X[age>cutpoint,1]+X[age>cutpoint,3]
y=mu+rnorm(100)
bestcut <- best.cut.node(X, y, Xsplit, split = "linear",
           glmnetParList=list(lambda = 0))
```

`defaults` *Default values passed to RotMat**

Description

Given the parameter list and the categorical map this function populates the values of the parameter list accoding to our 'best' known general use case parameters.

Usage

```
defaults(
  paramList,
  split = "entropy",
  dimX = NULL,
  weights = NULL,
  catLabel = NULL
)
```

Arguments

`paramList`	A list (possibly empty), to be populated with a set of default values to be passed to a `RotMat*` function.
`split`	The criterion used for splitting the variable. 'gini': gini impurity index (classification, default), 'entropy': information gain (classification) or 'mse': mean square error (regression).
`dimX`	An integer denoting the number of columns in the design matrix X.
`weights`	A vector of length same as `data` that are positive weights.(default NULL)
`catLabel`	A category labels of class `list` in predictors. (default NULL, for details see Examples of `ODT`)

Value

Default parameters of the RotMat* function.

- `dimX` An integer denoting the number of columns in the design matrix X.
- `dimProj` Number of variables to be projected, default `dimProj="Rand"`: random from 1 to ncol(X).
- `numProj` the number of projection directions.(default `ceiling(sqrt(dimX))`)
- `catLabel` A category labels of class `list` in prediction variables, for details see Examples of `ODRF`.
- `weights` A vector of length same as `data` that are positive weights.(default NULL)
- `lambda` Parameter of the Poisson distribution (default 1).
- `sparsity` A real number in $(0, 1)$ that specifies the distribution of non-zero elements in the random matrix. When `sparsity="pois"` means that non-zero elements are generated by the p(`lambda`) Poisson distribution.
- `prob` A probability $\in (0, 1)$ used for sampling from.
- `randDist` Parameter of the Poisson distribution (default 1).
- `split` The criterion used for splitting the variable. 'gini': gini impurity index (classification, default), 'entropy': information gain (classification) or 'mse': mean square error (regression).
- `model` Model for projection pursuit. (see `PPO`)

See Also

`RotMatPPO` `RotMatRand` `RotMatRF` `RotMatMake`

Examples

```
set.seed(1)
paramList <- list(dimX = 8, numProj = 3, sparsity = 0.25, prob = 0.5)
(paramList <- defaults(paramList, split = "entropy"))
```

ODBT *Classification and Regression using the Ensemble of ODT-based Boosting Trees*

Description

We use ODT as the basic tree model (base learner). To improve the performance of a boosting tree, we apply the feature bagging in this process, in the same way as the random forest. Our final estimator is called the ensemble of ODT-based boosting trees, denoted by ODBT, is the average of many boosting trees.

Usage

```
ODBT(X, ...)

## S3 method for class 'formula'
ODBT(
  formula,
  data = NULL,
  Xnew = NULL,
  type = "auto",
  model = c("ODT", "rpart", "rpart.cpp")[1],
  TreeRotate = TRUE,
  max.terms = 30,
  NodeRotateFun = "RotMatRF",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  ratOOB = 0.368,
  parallel = TRUE,
  numCores = Inf,
  MaxDepth = Inf,
  numNode = Inf,
 MinLeaf = ceiling(sqrt(ifelse(replacement, 1, 1 - ratOOB) * ifelse(is.null(data),
    length(eval(formula[[2]])), nrow(data)))/3),
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
```

```
  Xcat = 0,
  Xscale = "No",
  ...
)

## Default S3 method:
ODBT(
  X,
  y,
  Xnew = NULL,
  type = "auto",
  model = c("ODT", "rpart", "rpart.cpp")[1],
  TreeRotate = TRUE,
  max.terms = 30,
  NodeRotateFun = "RotMatRF",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  ratOOB = 0.368,
  parallel = TRUE,
  numCores = Inf,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = ceiling(sqrt(ifelse(replacement, 1, 1 - ratOOB) * length(y))/3),
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
  Xcat = 0,
  Xscale = "No",
  ...
)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
...	Optional parameters to be passed to the low level function.
formula	Object of class formula with a response describing the model to fit. If this is a data frame, it is taken as the model frame. (see model.frame)
data	Training data of class data.frame containing variables named in the formula. If data is missing it is obtained from the current environment by formula.
Xnew	An n by d numeric matrix (preferable) or data frame containing predictors for the new data.
type	Use ODBT for classification ("class") or regression ("reg").'auto' (default): If the response in data or y is a factor, "class" is used, otherwise regression is assumed.

`model` The basic tree model for boosting. We offer three options: "ODT" (default), "rpart," and "rpart.cpp" (improved "rpart")..

`TreeRotate` If or not to rotate the training data with the rotation matrix estimated by logistic regression before building the tree (default TRUE).

`max.terms` The maximum number of iterations for boosting trees.

`NodeRotateFun` Name of the function of class `character` that implements a linear combination of predictors in the split node. including

- "RotMatPPO": projection pursuit optimization model (`PPO`), see `RotMatPPO` (default, model="PPR").
- "RotMatRF": single feature similar to Random Forest, see `RotMatRF`.
- "RotMatRand": random rotation, see `RotMatRand`.
- "RotMatMake": users can define this function, for details see `RotMatMake`.

`FunDir` The path to the `function` of the user-defined `NodeRotateFun` (default current working directory).

`paramList` List of parameters used by the functions `NodeRotateFun`. If left unchanged, default values will be used, for details see `defaults`.

`ntrees` The number of trees in the forest (default 100).

`storeOOB` If TRUE then the samples omitted during the creation of a tree are stored as part of the tree (default TRUE).

`replacement` if TRUE then n samples are chosen, with replacement, from training data (default TRUE).

`stratify` If TRUE then class sample proportions are maintained during the random sampling. Ignored if replacement = FALSE (default TRUE).

`ratOOB` Ratio of 'out-of-bag' (default 1/3).

`parallel` Parallel computing or not (default TRUE).

`numCores` Number of cores to be used for parallel computing (default `Inf`).

`MaxDepth` The maximum depth of the tree (default `Inf`).

`numNode` Number of nodes that can be used by the tree (default `Inf`).

`MinLeaf` Minimal node size (Default 5).

`subset` An index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)

`weights` Vector of non-negative observational weights; fractional weights are allowed (default NULL).

`na.action` A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)

`catLabel` A category labels of class `list` in predictors. (default NULL, for details see Examples)

`Xcat` A class `vector` is used to indicate which predictor is the categorical variable, the default `Xcat`=0 means that no special treatment is given to category variables. When Xcat=NULL, the predictor x that satisfies the condition (length(unique(x))<10) & (n>20) is judged to be a category variable.

Xscale	Predictor standardization methods. " Min-max" (default), "Quantile", "No" denote Min-max transformation, Quantile transformation and No transformation respectively.
y	A response vector of length n.

Value

An object of class ODBT Containing a list components:

- call: The original call to ODBT.
- terms: An object of class c("terms", "formula") (see terms.object) summarizing the formula. Used by various methods, but typically not of direct relevance to users.
- ppTrees: Each tree used to build the forest.
  - oobErr: 'out-of-bag' error for tree, misclassification rate (MR) for classification or mean square error (MSE) for regression.
  - oobIndex: Which training data to use as 'out-of-bag'.
  - oobPred: Predicted value for 'out-of-bag'.
  - other: For other tree related values ODT.
- oobErr: 'out-of-bag' error for forest, misclassification rate (MR) for classification or mean square error (MSE) for regression.
- oobConfusionMat: 'out-of-bag' confusion matrix for forest.
- split, Levels and NodeRotateFun are important parameters for building the tree.
- paramList: Parameters in a named list to be used by NodeRotateFun.
- data: The list of data related parameters used to build the forest.
- tree: The list of tree related parameters used to build the tree.
- forest: The list of forest related parameters used to build the forest.
- results: The prediction results for new data Xnew using ODBT.

Author(s)

Yu Liu and Yingcun Xia

References

Zhan, H., Liu, Y., & Xia, Y. (2024). Consistency of Oblique Decision Tree and its Boosting and Random Forest. arXiv preprint arXiv:2211.12653.

Tomita, T. M., Browne, J., Shen, C., Chung, J., Patsolic, J. L., Falk, B., ... & Vogelstein, J. T. (2020). Sparse projection oblique randomer forests. Journal of machine learning research, 21(104).

See Also

ODT

Examples

```
# Classification with Oblique Decision Tree.
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODBT(varieties_of_wheat ~ ., train_data, test_data[, -8],model="rpart",
type = "class", parallel = FALSE, NodeRotateFun = "RotMatRF")
pred <- forest$results$prediction
# classification error
(mean(pred != test_data[, 8]))
forest <- ODBT(varieties_of_wheat ~ ., train_data, test_data[, -8],model="rpart.cpp",
type = "class", parallel = FALSE, NodeRotateFun = "RotMatRF")
pred <- forest$results$prediction
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Randome Forest.
data(body_fat)
set.seed(221212)
train <- sample(1:252, 80)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
# To use ODT as the basic tree model for boosting, you need to set
#the parameters model = "ODT" and NodeRotateFun = "RotMatPPO".
forest <- ODBT(Density ~ ., train_data, test_data[, -1],
  type = "reg",parallel = FALSE, model="ODT",
  NodeRotateFun = "RotMatPPO")
pred <- forest$results$prediction
# estimation error
mean((pred - test_data[, 1])^2)
forest <- ODBT(Density ~ ., train_data, test_data[, -1],
  type = "reg", parallel = FALSE,model="rpart.cpp",
  NodeRotateFun = "RotMatRF")
pred <- forest$results$prediction
# estimation error
mean((pred - test_data[, 1])^2)
```

ODRF *Classification and Regression using Oblique Decision Random Forest*

Description

Classification and regression implemented by the oblique decision random forest. ODRF usually produces more accurate predictions than RF, but needs longer computation time.

Usage

```
ODRF(X, ...)

## S3 method for class 'formula'
ODRF(
  formula,
  data = NULL,
  split = "auto",
  lambda = "log",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  ratOOB = 1/3,
  parallel = TRUE,
  numCores = Inf,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 5,
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
  Xcat = 0,
  Xscale = "Min-max",
  TreeRandRotate = FALSE,
  ...
)

## Default S3 method:
ODRF(
  X,
  y,
  split = "auto",
  lambda = "log",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  ratOOB = 1/3,
  parallel = TRUE,
  numCores = Inf,
```

```
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 5,
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
  Xcat = 0,
  Xscale = "Min-max",
  TreeRandRotate = FALSE,
  ...
)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
...	Optional parameters to be passed to the low level function.
formula	Object of class formula with a response describing the model to fit. If this is a data frame, it is taken as the model frame. (see model.frame)
data	Training data of class data.frame containing variables named in the formula. If data is missing it is obtained from the current environment by formula.
split	The criterion used for splitting the nodes. "entropy": information gain and "gini": gini impurity index for classification; "mse": mean square error for regression; 'auto' (default): If the response in data or y is a factor, "gini" is used, otherwise regression is assumed.
lambda	The argument of split is used to determine the penalty level of the partition criterion. Three options are provided including, lambda=0: no penalty; lambda=2: AIC penalty; lambda='log' (Default): BIC penalty. In Addition, lambda can be any value from 0 to n (training set size).
NodeRotateFun	Name of the function of class character that implements a linear combination of predictors in the split node. including • "RotMatPPO": projection pursuit optimization model (PPO), see RotMatPPO (default, model="PPR"). • "RotMatRF": single feature similar to Random Forest, see RotMatRF. • "RotMatRand": random rotation, see RotMatRand. • "RotMatMake": users can define this function, for details see RotMatMake.
FunDir	The path to the function of the user-defined NodeRotateFun (default current working directory).
paramList	List of parameters used by the functions NodeRotateFun. If left unchanged, default values will be used, for details see defaults.
ntrees	The number of trees in the forest (default 100).
storeOOB	If TRUE then the samples omitted during the creation of a tree are stored as part of the tree (default TRUE).
replacement	if TRUE then n samples are chosen, with replacement, from training data (default TRUE).

stratify	If TRUE then class sample proportions are maintained during the random sampling. Ignored if replacement = FALSE (default TRUE).
ratOOB	Ratio of 'out-of-bag' (default 1/3).
parallel	Parallel computing or not (default TRUE).
numCores	Number of cores to be used for parallel computing (default Inf).
MaxDepth	The maximum depth of the tree (default Inf).
numNode	Number of nodes that can be used by the tree (default Inf).
MinLeaf	Minimal node size (Default 5).
subset	An index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)
weights	Vector of non-negative observational weights; fractional weights are allowed (default NULL).
na.action	A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)
catLabel	A category labels of class list in predictors. (default NULL, for details see Examples)
Xcat	A class vector is used to indicate which predictor is the categorical variable. The default Xcat=0 means that no special treatment is given to category variables. When Xcat=NULL, the predictor x that satisfies the condition "(length(table(x))<10) & (length(x)>20)" is judged to be a category variable.
Xscale	Predictor standardization methods. " Min-max" (default), "Quantile", "No" denote Min-max transformation, Quantile transformation and No transformation respectively.
TreeRandRotate	If or not to randomly rotate the training data before building the tree (default FALSE, see RandRot).
y	A response vector of length n.

Value

An object of class ODRF Containing a list components:

- call: The original call to ODRF.
- terms: An object of class c("terms", "formula") (see terms.object) summarizing the formula. Used by various methods, but typically not of direct relevance to users.
- split, Levels and NodeRotateFun are important parameters for building the tree.
- predicted: the predicted values of the training data based on out-of-bag samples.
- paramList: Parameters in a named list to be used by NodeRotateFun.
- oobErr: 'out-of-bag' error for forest, misclassification rate (MR) for classification or mean square error (MSE) for regression.
- oobConfusionMat: 'out-of-bag' confusion matrix for forest.
- structure: Each tree structure used to build the forest.
  - oobErr: 'out-of-bag' error for tree, misclassification rate (MR) for classification or mean square error (MSE) for regression.

  - oobIndex: Which training data to use as 'out-of-bag'.
  - oobPred: Predicted value for 'out-of-bag'.
  - others: Same tree structure return value as ODT.
- data: The list of data related parameters used to build the forest.
- tree: The list of tree related parameters used to build the tree.
- forest: The list of forest related parameters used to build the forest.

Author(s)

Yu Liu and Yingcun Xia

References

Zhan, H., Liu, Y., & Xia, Y. (2022). Consistency of The Oblique Decision Tree and Its Random Forest. arXiv preprint arXiv:2211.12653.

Tomita, T. M., Browne, J., Shen, C., Chung, J., Patsolic, J. L., Falk, B., ... & Vogelstein, J. T. (2020). Sparse projection oblique randomer forests. Journal of machine learning research, 21(104).

See Also

online.ODRF prune.ODRF predict.ODRF print.ODRF Accuracy VarImp

Examples

```
# Classification with Oblique Decision Randome Forest.
data(seeds)
set.seed(221212)
train <- sample(1:209, 80)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., train_data,
  split = "entropy",parallel = FALSE, ntrees = 50
)
pred <- predict(forest, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Randome Forest.
data(body_fat)
set.seed(221212)
train <- sample(1:252, 80)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
forest <- ODRF(Density ~ ., train_data,
  split = "mse", parallel = FALSE,
  NodeRotateFun = "RotMatPPO", paramList = list(model = "Log", dimProj = "Rand")
)
pred <- predict(forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

```
### Train ODRF on one-of-K encoded categorical data ###
# Note that the category variable must be placed at the beginning of the predictor X
# as in the following example.
set.seed(22)
Xcol1 <- sample(c("A", "B", "C"), 100, replace = TRUE)
Xcol2 <- sample(c("1", "2", "3", "4", "5"), 100, replace = TRUE)
Xcon <- matrix(rnorm(100 * 3), 100, 3)
X <- data.frame(Xcol1, Xcol2, Xcon)
Xcat <- c(1, 2)
catLabel <- NULL
y <- as.factor(sample(c(0, 1), 100, replace = TRUE))

forest <- ODRF(y ~ X, split = "entropy", Xcat = NULL, parallel = FALSE)

head(X)
#>   Xcol1 Xcol2          X1         X2          X3
#> 1     B     5 -0.04178453  2.3962339 -0.01443979
#> 2     A     4 -1.66084623 -0.4397486  0.57251733
#> 3     B     2 -0.57973333 -0.2878683  1.24475578
#> 4     B     1 -0.82075051  1.3702900  0.01716528
#> 5     C     5 -0.76337897 -0.9620213  0.25846351
#> 6     A     5 -0.37720294 -0.1853976  1.04872159

# one-of-K encode each categorical feature and store in X1
numCat <- apply(X[, Xcat, drop = FALSE], 2, function(x) length(unique(x)))
# initialize training data matrix X1
X1 <- matrix(0, nrow = nrow(X), ncol = sum(numCat))
catLabel <- vector("list", length(Xcat))
names(catLabel) <- colnames(X)[Xcat]
col.idx <- 0L
# convert categorical feature to K dummy variables
for (j in seq_along(Xcat)) {
  catMap <- (col.idx + 1):(col.idx + numCat[j])
  catLabel[[j]] <- levels(as.factor(X[, Xcat[j]]))
  X1[, catMap] <- (matrix(X[, Xcat[j]], nrow(X), numCat[j]) ==
    matrix(catLabel[[j]], nrow(X), numCat[j], byrow = TRUE)) + 0
  col.idx <- col.idx + numCat[j]
}
X <- cbind(X1, X[, -Xcat])
colnames(X) <- c(paste(rep(seq_along(numCat), numCat), unlist(catLabel),
  sep = "."
), "X1", "X2", "X3")

# Print the result after processing of category variables.
head(X)
#>   1.A 1.B 1.C 2.1 2.2 2.3 2.4 2.5          X1         X2          X3
#> 1   0   1   0   0   0   0   0   1 -0.04178453  2.3962339 -0.01443979
#> 2   1   0   0   0   0   0   1   0 -1.66084623 -0.4397486  0.57251733
#> 3   0   1   0   0   1   0   0   0 -0.57973333 -0.2878683  1.24475578
#> 4   0   1   0   1   0   0   0   0 -0.82075051  1.3702900  0.01716528
#> 5   0   0   1   0   0   0   0   1 -0.76337897 -0.9620213  0.25846351
```

```
#> 6   1   0   0   0   0   0   0   1 -0.37720294 -0.1853976  1.04872159
catLabel
#> $Xcol1
#> [1] "A" "B" "C"
#>
#> $Xcol2
#> [1] "1" "2" "3" "4" "5"


forest <- ODRF(X, y,
  split = "gini", Xcat = c(1, 2),
  catLabel = catLabel, parallel = FALSE
)
```

ODT *Classification and Regression with Oblique Decision Tree*

Description

Classification and regression using an oblique decision tree (ODT) in which each node is split by a linear combination of predictors. Different methods are provided for selecting the linear combinations, while the splitting values are chosen by one of three criteria.

Usage

```
ODT(X, ...)

## S3 method for class 'formula'
ODT(
  formula,
  data = NULL,
  Xsplit = NULL,
  split = "auto",
  lambda = "log",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  glmnetParList = NULL,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 10,
  Levels = NULL,
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
```

```
  Xcat = 0,
  Xscale = "Min-max",
  TreeRandRotate = FALSE,
  ...
)

## Default S3 method:
ODT(
  X,
  y,
  Xsplit = NULL,
  split = "auto",
  lambda = "log",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  glmnetParList = NULL,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 10,
  Levels = NULL,
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
  Xcat = 0,
  Xscale = "Min-max",
  TreeRandRotate = FALSE,
  ...
)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
...	Optional parameters to be passed to the low level function.
formula	Object of class formula with a response describing the model to fit. If this is a data frame, it is taken as the model frame. (see model.frame)
data	Training data of class data.frame containing variables named in the formula. If data is missing it is obtained from the current environment by formula.
Xsplit	Splitting variables used to construct linear model trees. The default value is NULL and is only valid when split="linear".
split	The criterion used for splitting the nodes. "entropy": information gain and "gini": gini impurity index for classification; "mse": mean square error for regression; "linear": mean square error for linear model. 'auto' (default): If the response in data or y is a factor, "gini" is used, otherwise "mse" is assumed.
lambda	The argument of split is used to determine the penalty level of the partition criterion. Three options are provided including, lambda=0: no penalty; lambda=2:

AIC penalty; `lambda='log'` (Default): BIC penalty. In Addition, lambda can be any value from 0 to n (training set size).

`NodeRotateFun` Name of the function of class `character` that implements a linear combination of predictors in the split node. including

- "RotMatPPO": projection pursuit optimization model (`PPO`), see `RotMatPPO` (default, model="PPR").
- "RotMatRF": single feature similar to CART, see `RotMatRF`.
- "RotMatRand": random rotation, see `RotMatRand`.
- "RotMatMake": users can define this function, for details see `RotMatMake`.

`FunDir` The path to the `function` of the user-defined `NodeRotateFun` (default current working directory).

`paramList` List of parameters used by the functions `NodeRotateFun`. If left unchanged, default values will be used, for details see `defaults`.

`glmnetParList` List of parameters used by the functions `glmnet` and `cv.glmnet` in package `glmnet`. `glmnetParList=list(lambda = 0)` is Ordinary Least Squares (OLS) regression, `glmnetParList=list(family = "gaussian")` (default) is regression model and `glmnetParList=list(family = "binomial" or "multinomial")` is classification model. If left unchanged, default values will be used, for details see `glmnet` and `cv.glmnet`.

`MaxDepth` The maximum depth of the tree (default `Inf`).

`numNode` Number of nodes that can be used by the tree (default `Inf`).

`MinLeaf` Minimal node size (Default 10).

`Levels` The category label of the response variable when `split` is not equal to 'mse'.

`subset` An index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)

`weights` Vector of non-negative observational weights; fractional weights are allowed (default NULL).

`na.action` A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)

`catLabel` A category labels of class `list` in predictors. (default NULL, for details see Examples)

`Xcat` A class `vector` is used to indicate which predictor is the categorical variable. The default Xcat=0 means that no special treatment is given to category variables. When Xcat=NULL, the predictor x that satisfies the condition "`(length(table(x))<10) & (length(x)>20)`" is judged to be a category variable.

`Xscale` Predictor standardization methods. " Min-max" (default), "Quantile", "No" denote Min-max transformation, Quantile transformation and No transformation respectively.

`TreeRandRotate` If or not to randomly rotate the training data before building the tree (default FALSE, see `RandRot`).

`y` A response vector of length n.

Value

An object of class ODT containing a list of components::

- `call`: The original call to ODT.
- `terms`: An object of class `c("terms", "formula")` (see `terms.object`) summarizing the formula. Used by various methods, but typically not of direct relevance to users.
- `split`, `Levels` and `NodeRotateFun` are important parameters for building the tree.
- `predicted`: the predicted values of the training data.
- `projections`: Projection direction for each split node.
- `paramList`: Parameters in a named list to be used by `NodeRotateFun`.
- `data`: The list of data related parameters used to build the tree.
- `tree`: The list of tree related parameters used to build the tree.
- `structure`: A set of tree structure data records.
  - `nodeRotaMat`: Record the split variables (first column), split node serial number (second column) and rotation direction (third column) for each node. (The first column and the third column are 0 means leaf nodes)
  - `nodeNumLabel`: Record each leaf node's category for classification or predicted value for regression (second column is data size). (Each column is 0 means it is not a leaf node)
  - `nodeCutValue`: Record the split point of each node. (0 means leaf nodes)
  - `nodeCutIndex`: Record the index values of the partitioning variables selected based on the partition criterion `split`.
  - `childNode`: Record the number of child nodes after each splitting.
  - `nodeDepth`: Record the depth of the tree where each node is located.
  - `nodeIndex`: Record the indices of the data used in each node.
  - `glmnetFit`: Record the model fitted by function `glmnet` used in each node.

Author(s)

Yu Liu and Yingcun Xia

References

Zhan, H., Liu, Y., & Xia, Y. (2022). Consistency of The Oblique Decision Tree and Its Random Forest. arXiv preprint arXiv:2211.12653.

See Also

`online.ODT` `prune.ODT` `as.party` `predict.ODT` `print.ODT` `plot.ODT` `plot_ODT_depth`

Examples

```
# Classification with Oblique Decision Tree.
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
```

```
test_data <- data.frame(seeds[-train, ])
tree <- ODT(varieties_of_wheat ~ ., train_data, split = "entropy")
pred <- predict(tree, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Tree.
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
tree <- ODT(Density ~ ., train_data,
  split = "mse",
  NodeRotateFun = "RotMatPPO", paramList = list(model = "Log", dimProj = "Rand")
)
pred <- predict(tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)

# Use "Z" as the splitting variable to build a linear model tree for "X" and "y".
set.seed(10)
cutpoint=50
X=matrix(rnorm(100*10),100,10)
age=sample(seq(20,80),100,replace = TRUE)
height=sample(seq(50,200),100,replace = TRUE)
weight=sample(seq(5,150),100,replace = TRUE)
Z=cbind(age=age,height=height,weight=weight)
mu=rep(0,100)
mu[age<=cutpoint]=X[age<=cutpoint,1]+X[age<=cutpoint,2]
mu[age>cutpoint]=X[age>cutpoint,1]+X[age>cutpoint,3]
y=mu+rnorm(100)
# Regression model tree
my.tree <- ODT(X=X, y=y, Xsplit=Z, split = "linear", lambda = 0,
NodeRotateFun = "RotMatRF",
glmnetParList=list(lambda = 0, family = "gaussian"))
pred <- predict(my.tree, X, Xsplit=Z)
# fitting error
mean((pred - y)^2)
mean((my.tree$predicted - y)^2)
# Classification model tree
y1 = (y>0)*1
my.tree <- ODT(X=X, y=y1, Xsplit=Z, split = "linear",lambda = 0,
               NodeRotateFun = "RotMatRF",MinLeaf = 10, MaxDepth = 5,
               glmnetParList=list(family = "binomial"))
(class <- predict(my.tree, X, Xsplit=Z, type="pred"))
(prob <- predict(my.tree, X, Xsplit=Z, type="prob"))

# Projection analysis of the oblique decision tree.
data(iris)
tree <- ODT(Species ~ ., data = iris, split="gini",
            paramList = list(model = "PPR", numProj = 1))
print(round(tree[["projections"]],3))
```

```
### Train ODT on one-of-K encoded categorical data ###
# Note that the category variable must be placed at the beginning of the predictor X
# as in the following example.
set.seed(22)
Xcol1 <- sample(c("A", "B", "C"), 100, replace = TRUE)
Xcol2 <- sample(c("1", "2", "3", "4", "5"), 100, replace = TRUE)
Xcon <- matrix(rnorm(100 * 3), 100, 3)
X <- data.frame(Xcol1, Xcol2, Xcon)
Xcat <- c(1, 2)
catLabel <- NULL
y <- as.factor(sample(c(0, 1), 100, replace = TRUE))
tree <- ODT(X, y, split = "entropy", Xcat = NULL)
head(X)
#>   Xcol1 Xcol2          X1         X2          X3
#> 1     B     5 -0.04178453  2.3962339 -0.01443979
#> 2     A     4 -1.66084623 -0.4397486  0.57251733
#> 3     B     2 -0.57973333 -0.2878683  1.24475578
#> 4     B     1 -0.82075051  1.3702900  0.01716528
#> 5     C     5 -0.76337897 -0.9620213  0.25846351
#> 6     A     5 -0.37720294 -0.1853976  1.04872159

# one-of-K encode each categorical feature and store in X1
numCat <- apply(X[, Xcat, drop = FALSE], 2, function(x) length(unique(x)))
# initialize training data matrix X
X1 <- matrix(0, nrow = nrow(X), ncol = sum(numCat))
catLabel <- vector("list", length(Xcat))
names(catLabel) <- colnames(X)[Xcat]
col.idx <- 0L
# convert categorical feature to K dummy variables
for (j in seq_along(Xcat)) {
  catMap <- (col.idx + 1):(col.idx + numCat[j])
  catLabel[[j]] <- levels(as.factor(X[, Xcat[j]]))
  X1[, catMap] <- (matrix(X[, Xcat[j]], nrow(X), numCat[j]) ==
    matrix(catLabel[[j]], nrow(X), numCat[j], byrow = TRUE)) + 0
  col.idx <- col.idx + numCat[j]
}
X <- cbind(X1, X[, -Xcat])
colnames(X) <- c(paste(rep(seq_along(numCat), numCat), unlist(catLabel),
  sep = ".")
), "X1", "X2", "X3")

# Print the result after processing of category variables.
head(X)
#>   1.A 1.B 1.C 2.1 2.2 2.3 2.4 2.5          X1         X2          X3
#> 1   0   1   0   0   0   0   0   1 -0.04178453  2.3962339 -0.01443979
#> 2   1   0   0   0   0   0   1   0 -1.66084623 -0.4397486  0.57251733
#> 3   0   1   0   0   1   0   0   0 -0.57973333 -0.2878683  1.24475578
#> 4   0   1   0   1   0   0   0   0 -0.82075051  1.3702900  0.01716528
#> 5   0   0   1   0   0   0   0   1 -0.76337897 -0.9620213  0.25846351
#> 6   1   0   0   0   0   0   0   1 -0.37720294 -0.1853976  1.04872159
catLabel
#> $Xcol1
```

```
#> [1] "A" "B" "C"
#>
#> $Xcol2
#> [1] "1" "2" "3" "4" "5"

tree <- ODT(X, y, split = "gini", Xcat = c(1, 2), catLabel = catLabel,NodeRotateFun = "RotMatRF")
```

`online.ODRF` *using new training data to update an existing* `ODRF`.

Description

Update existing `ODRF` using new data to improve the model.

Usage

```
## S3 method for class 'ODRF'
online(obj, X, y, weights = NULL, MaxDepth = Inf, ...)
```

Arguments

`obj`	An object of class `ODRF`.
`X`	An new n by d numeric matrix (preferable) or data frame used to update the object of class `ODRF`.
`y`	A new response vector of length n used to update the object of class `ODRF`.
`weights`	A vector of non-negative observational weights; fractional weights are allowed (default NULL).
`MaxDepth`	The maximum depth of the tree (default `Inf`).
`...`	Optional parameters to be passed to the low level function.

Value

The same result as `ODRF`.

See Also

`ODRF` `prune.ODRF` `online.ODT`

Examples

```
# Classification with Oblique Decision Random Forest
data(seeds)
set.seed(221212)
train <- sample(1:209, 80)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
```

```
index <- seq(floor(nrow(train_data) / 2))
forest <- ODRF(varieties_of_wheat ~ ., train_data[index, ],
  split = "gini", parallel = FALSE, ntrees = 50
)
online_forest <- online(forest, train_data[-index, -8], train_data[-index, 8])
pred <- predict(online_forest, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Random Forest

data(body_fat)
set.seed(221212)
train <- sample(1:252, 80)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
index <- seq(floor(nrow(train_data) / 2))
forest <- ODRF(Density ~ ., train_data[index, ],
  split = "mse", parallel = FALSE
)
online_forest <- online(
  forest, train_data[-index, -1],
  train_data[-index, 1]
)
pred <- predict(online_forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

`online.ODT` *using new training data to update an existing* `ODT`.

Description

Update existing `ODT` using new data to improve the model.

Usage

```
## S3 method for class 'ODT'
online(obj, X = NULL, y = NULL, weights = NULL, MaxDepth = Inf, ...)
```

Arguments

`obj`	an object of class `ODT`.
`X`	An new n by d numeric matrix (preferable) or data frame used to update the object of class `ODT`.
`y`	A new response vector of length n used to update the object of class `ODT`.

weights	Vector of non-negative observational weights; fractional weights are allowed (default NULL).
MaxDepth	The maximum depth of the tree (default Inf).
...	optional parameters to be passed to the low level function.

Value

The same result as ODT.

See Also

ODT prune.ODT online.ODRF

Examples

```
# Classification with Oblique Decision Tree
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
index <- seq(floor(nrow(train_data) / 2))
tree <- ODT(varieties_of_wheat ~ ., train_data[index, ], split = "gini")
online_tree <- online(tree, train_data[-index, -8], train_data[-index, 8])
pred <- predict(online_tree, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Tree
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
index <- seq(floor(nrow(train_data) / 2))
tree <- ODT(Density ~ ., train_data[index, ], split = "mse")
online_tree <- online(tree, train_data[-index, -1], train_data[-index, 1])
pred <- predict(online_tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

---

plot.Accuracy *plot method for* Accuracy *objects*

Description

Draw the error graph of class ODRF at different tree sizes.

Usage

```
## S3 method for class 'Accuracy'
plot(x, lty = 1, digits = NULL, main = NULL, ...)
```

Arguments

x	Object of class Accuracy.
lty	A vector of line types, see par.
digits	Integer indicating the number of decimal places (round) or significant digits (signif) to be used.
main	main title of the plot.
...	Arguments to be passed to methods.

Value

OOB error and test error, misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

ODRF Accuracy

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 80)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, split = "gini",
parallel = FALSE, ntrees = 30)
(error <- Accuracy(forest, train_data, test_data))
plot(error)
```

---

plot.ODT *to plot an oblique decision tree*

Description

Draw oblique decision tree with tree structure. It is modified from a function in PPtreeViz library.

Usage

```
## S3 method for class 'ODT'
plot(x, font.size = 17, width.size = 1, xadj = 0, main = NULL, sub = NULL, ...)
```

Arguments

x	An object of class ODT.
font.size	Font size of plot
width.size	Size of eclipse in each node.
xadj	The size of the left and right movement.
main	main title
sub	sub title
...	Arguments to be passed to methods.

Value

Tree Structure.

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software.

See Also

ODT as.party plot_ODT_depth

Examples

```
data(iris)
tree <- ODT(Species ~ ., data = iris, split = "gini")
plot(tree)
```

---

plot.prune.ODT *to plot pruned oblique decision tree*

Description

Plot the error graph of the pruned oblique decision tree at different split nodes.

Usage

```
## S3 method for class 'prune.ODT'
plot(x, position = "topleft", digits = NULL, main = NULL, ...)
```

Arguments

x	An object of class prune.ODT.
position	Position of the curve label, including "topleft" (default), "bottomright", "bottom", "bottomleft", "left", "top", "topright", "right" and "center".
digits	Integer indicating the number of decimal places (round) or significant digits (signif) to be used.
main	main title
...	Arguments to be passed to methods.

Value

The leftmost value of the horizontal axis indicates the tree without pruning, while the rightmost value indicates the data without splitting and using the average value as the predicted value.

See Also

ODT prune.ODT

Examples

```
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])

tree <- ODT(Density ~ ., train_data, split = "mse")
prune_tree <- prune(tree, test_data[, -1], test_data[, 1])
# Plot pruned oblique decision tree structure (default)
plot(prune_tree)
# Plot the error graph of the pruned oblique decision tree.
class(prune_tree) <- "prune.ODT"
plot(prune_tree)
```

---

plot.VarImp *Variable Importance Plot*

Description

Dotchart of variable importance as measured by an Oblique Decision Random Forest.

Usage

```
## S3 method for class 'VarImp'
plot(x, nvar = min(30, nrow(x$varImp)), digits = NULL, main = NULL, ...)
```

Arguments

x	An object of class VarImp.
nvar	number of variables to show.
digits	Integer indicating the number of decimal places (round) or significant digits (signif) to be used.
main	plot title.
...	Arguments to be passed to methods.

Value

The horizontal axis is the increased error of ODRF after replacing the variable, the larger the increased error the more important the variable is.

See Also

ODRF VarImp

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
forest <- ODRF(train_data[, -1], train_data[, 1], split = "gini",
  parallel = FALSE)
varimp <- VarImp(forest, train_data[, -1], train_data[, 1])
plot(varimp)
```

---

plot_ODT_depth *plot oblique decision tree depth*

Description

Draw the error graph of class ODT at different depths.

Usage

```
plot_ODT_depth(
  formula,
  data = NULL,
  newdata = NULL,
  split = "gini",
  NodeRotateFun = "RotMatPPO",
  paramList = NULL,
  digits = NULL,
  main = NULL,
  ...
)
```

Arguments

formula	Object of class formula with a response describing the model to fit. If this is a data frame, it is taken as the model frame. (see model.frame)
data	Training data of class data.frame in ODT used to calculate the OOB error.
newdata	A data frame or matrix containing new data is used to calculate the test error. If it is missing, then it is replaced by data.
split	The criterion used for splitting the variable. 'gini': gini impurity index (classification, default), 'entropy': information gain (classification) or 'mse': mean square error (regression).
NodeRotateFun	Name of the function of class character that implements a linear combination of predictors in the split node. including • "RotMatPPO": projection pursuit optimization model (PPO), see RotMatPPO (default, model="PPR"). • "RotMatRF": single feature similar to Random Forest, see RotMatRF. • "RotMatRand": random rotation, see RotMatRand. • "RotMatMake": Users can define this function, for details see RotMatMake.
paramList	List of parameters used by the functions NodeRotateFun. If left unchanged, default values will be used, for details see defaults.
digits	Integer indicating the number of decimal places (round) or significant digits (signif) to be used.
main	main title
...	Arguments to be passed to methods.

Value

OOB error and test error of newdata, misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

ODT plot.ODT

Examples

```
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
plot_ODT_depth(Density ~ ., train_data, test_data, split = "mse")
```

PPO *Projection Pursuit Optimization*

Description

Find the optimal projection using various projectin pursuit models.

Usage

```
PPO(X, y, model = "PPR", split = "gini", weights = NULL, ...)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
y	A response vector of length n.
model	Model for projection pursuit. • "PPR"(default): projection projection regression from ppr. When y is a category label, it is expanded to K binary features. • "Log": logistic based on nnet. • "Rand": The random projection generated from $\{-1, 1\}$. The following models can only be used for classification, i.e. the split must be "entropy" or 'gini'. • "LDA", "PDA", "Lr", "GINI", and "ENTROPY" from library PPtreeViz. • The following models based on Pursuit. – "holes": Holes index – "cm": Central Mass index – "holes": Holes index – "friedmantukey": Friedman Tukey index – "legendre": Legendre index – "laguerrefourier": Laguerre Fourier index – "hermite": Hermite index – "naturalhermite": Natural Hermite index – "kurtosismax": Maximum kurtosis index – "kurtosismin": Minimum kurtosis index – "moment": Moment index – "mf": MF index – "chi": Chi-square index
split	The criterion used for splitting the variable. 'gini': gini impurity index (classification, default), 'entropy': information gain (classification) or 'mse': mean square error (regression).
weights	Vector of non-negative observational weights; fractional weights are allowed (default NULL).
...	optional parameters to be passed to the low level function.

Value

Optimal projection direction.

References

Friedman, J. H., & Stuetzle, W. (1981). Projection pursuit regression. Journal of the American statistical Association, 76(376), 817-823.

Ripley, B. D. (1996) Pattern Recognition and Neural Networks. Cambridge.

Lee, YD, Cook, D., Park JW, and Lee, EK(2013) PPtree: Projection Pursuit Classification Tree, Electronic Journal of Statistics, 7:1369-1386.

Cook, D., Buja, A., Lee, E. K., & Wickham, H. (2008). Grand tours, projection pursuit guided tours, and manual controls. In Handbook of data visualization (pp. 295-314). Springer, Berlin, Heidelberg.

See Also

RotMatPPO

Examples

```
# classification
data(seeds)
(PP <- PPO(seeds[, 1:7], seeds[, 8], model = "Log", split = "entropy"))
(PP <- PPO(seeds[, 1:7], seeds[, 8], model = "PPR", split = "entropy"))
(PP <- PPO(seeds[, 1:7], seeds[, 8], model = "LDA", split = "entropy"))

# regression
data(body_fat)
(PP <- PPO(body_fat[, 2:15], body_fat[, 1], model = "Log", split = "mse"))
(PP <- PPO(body_fat[, 2:15], body_fat[, 1], model = "Rand", split = "mse"))
(PP <- PPO(body_fat[, 2:15], body_fat[, 1], model = "PPR", split = "mse"))
```

---

predict.ODRF *predict based on an ODRF object*

Description

Prediction of ODRF for an input matrix or data frame.

Usage

```
## S3 method for class 'ODRF'
predict(object, Xnew, type = "response", weight.tree = FALSE, ...)
```

Arguments

object	An object of class ODRF, the same created by the function ODRF.
Xnew	An n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features. Note that if there are NA values in the data 'Xnew', which will be replaced with the average value.
type	One of response, prob or tree, indicating the type of output: predicted values, matrix of class probabilities or predicted value for each tree.
weight.tree	Whether to weight the tree, if TRUE then use the out-of-bag error of the tree as the weight. (default FALSE)
...	Arguments to be passed to methods.

Value

A set of vectors in the following list:

- response: the predicted values of the new data.
- prob: matrix of class probabilities (one column for each class and one row for each input). If object$split is mse, a vector of tree weights is returned.
- tree: It is a matrix where each column is a prediction for each tree.

References

Zhan, H., Liu, Y., & Xia, Y. (2022). Consistency of The Oblique Decision Tree and Its Random Forest. arXiv preprint arXiv:2211.12653.

See Also

ODRF predict.ODT

Examples

```
# Classification with Oblique Decision Random Forest
data(seeds)
set.seed(221212)
train <- sample(1:209, 80)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., train_data,
  split = "entropy", parallel = FALSE,ntrees = 50
)
pred <- predict(forest, test_data[, -8], weight.tree = TRUE)
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Random Forest

data(body_fat)
set.seed(221212)
train <- sample(1:252, 80)
```

```
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
forest <- ODRF(Density ~ ., train_data, split = "mse", parallel = FALSE,
ntrees = 50, TreeRandRotate=TRUE)
pred <- predict(forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

`predict.ODT` *making predict based on ODT objects*

Description

Prediction of ODT for an input matrix or data frame.

Usage

```
## S3 method for class 'ODT'
predict(
  object,
  Xnew,
  Xsplit = NULL,
  type = c("pred", "leafnode", "prob")[1],
  ...
)
```

Arguments

`object`	An object of class ODT, the same as that created by the function `ODT`.
`Xnew`	An n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features. Note that if there are NA values in the data 'Xnew', which will be replaced with the average value.
`Xsplit`	Splitting variables used to construct linear model trees. The default value is NULL and is only valid when `split="linear"`.
`type`	Type of prediction required. Choosing `"pred"` (default) gives the prediction result, and choosing `"leafnode"` gives the leaf node sequence number that `Xnew` is partitioned into. For classification tasks, including classification trees (`split="gini"` or `"entropy"`) and linear classification models (`split="linear"` and `glmnetParList= list(family="binomial"` or `"multinomial")`). Setting `type="prob"` gives the prediction probabilities.
`...`	Arguments to be passed to methods.

Value

A vector of the following:

- pred: the prediced response of the new data.
- leafnode: the leaf node sequence number that the new data is partitioned.
- prob: the prediction probabilities for classification tasks.

References

Zhan, H., Liu, Y., & Xia, Y. (2022). Consistency of The Oblique Decision Tree and Its Random Forest. arXiv preprint arXiv:2211.12653.

See Also

ODT predict.ODRF

Examples

```
# Classification with Oblique Decision Tree.
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
tree <- ODT(varieties_of_wheat ~ ., train_data, split = "entropy")
pred <- predict(tree, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))
(prob=predict(tree, test_data[, -8],type = "prob"))

# Regression with Oblique Decision Tree.
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
tree <- ODT(Density ~ ., train_data, split = "mse")
pred <- predict(tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)

# Use "Z" as the splitting variable to build a linear model tree for "X" and "y".
set.seed(1)
n = 200
p = 10
q = 5
X = matrix(rnorm(n*p), n, p)
Z = matrix(rnorm(n*q), n, q)
y = (Z[,1] > 1)*(X[,1] - X[,2] + 2)  +
(Z[,1] < 1)*(Z[,2] > 0)*(X[,1] + X[,2] + 0) +
(Z[,1] < 1)*(Z[,2] < 0)*(X[,3] - 2)
```

```
my.tree <- ODT(X=X, y=y, Xsplit=Z, split = "linear",
               NodeRotateFun = "RotMatRF",MinLeaf = 10, MaxDepth = 5,
               glmnetParList=list(lambda = 0.1,family = "gaussian"))
(leafnode <- predict(my.tree, X, Xsplit=Z, type="leafnode"))

y1 = (y>0)*1
my.tree <- ODT(X=X, y=y1, Xsplit=Z, split = "linear",
               NodeRotateFun = "RotMatRF",MinLeaf = 10, MaxDepth = 5,
               glmnetParList=list(family = "binomial"))
(class <- predict(my.tree, X, Xsplit=Z, type="pred"))
(prob <- predict(my.tree, X, Xsplit=Z, type="prob"))

y2 = (y < -2.5)*1+(y>=-2.5&y<2.5)*2+(y>=2.5)*3
my.tree <- ODT(X=X, y=y2, Xsplit=Z, split = "linear",
               NodeRotateFun = "RotMatRF",MinLeaf = 10, MaxDepth = 5,
               glmnetParList=list(family = "multinomial"))
(prob <- predict(my.tree, X, Xsplit=Z, type="prob"))
```

`print.ODRF` *print ODRF*

Description

Print contents of ODRF object.

Usage

```
## S3 method for class 'ODRF'
print(x, ...)
```

Arguments

`x`	An object of class `ODRF`.
`...`	Arguments to be passed to methods.

Value

OOB error, misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

`ODRF`

Examples

```
data(iris)
forest <- ODRF(Species ~ ., data = iris, parallel = FALSE, ntrees = 50)
forest
```

`print.ODT` *print ODT result*

Description

Print the oblique decision tree structure.

Usage

```
## S3 method for class 'ODT'
print(x, projection = FALSE, cutvalue = FALSE, verbose = TRUE, ...)
```

Arguments

`x`	An object of class `ODT`.
`projection`	Print projection coefficients in each node if TRUE.
`cutvalue`	Print cutoff values in each node if TRUE.
`verbose`	Print if TRUE, no output if FALSE.
`...`	Arguments to be passed to methods.

Value

The oblique decision tree structure.

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software.

See Also

`ODT`

Examples

```
data(iris)
tree <- ODT(Species ~ ., data = iris)
tree
print(tree, projection = TRUE, cutvalue = TRUE)
```

prune.ODRF *Pruning of class* ODRF.

Description

Prune ODRF from bottom to top with test data based on prediction error.

Usage

```
## S3 method for class 'ODRF'
prune(obj, X, y, MaxDepth = 1, useOOB = TRUE, ...)
```

Arguments

obj	An object of class ODRF.
X	An n by d numeric matrix (preferable) or data frame is used to prune the object of class ODRF.
y	A response vector of length n.
MaxDepth	The maximum depth of the tree after pruning (Default 1).
useOOB	Whether to use OOB for pruning (Default TRUE). Note that when useOOB=TRUE, X and y must be the training data in ODRF.
...	Optional parameters to be passed to the low level function.

Value

An object of class ODRF and prune.ODRF.

- ppForest The same result as ODRF.
- pruneError Error of test data or OOB after each pruning in each tree, misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

ODRF online.ODRF prune.ODT

Examples

```
# Classification with Oblique Decision Random Forest
data(seeds)
set.seed(221212)
train <- sample(1:209, 80)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., train_data,
  split = "entropy", parallel = FALSE, ntrees = 50
)
prune_forest <- prune(forest, train_data[, -8], train_data[, 8])
```

```
pred <- predict(prune_forest, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Random Forest
data(body_fat)
set.seed(221212)
train <- sample(1:252,80)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
index <- seq(floor(nrow(train_data) / 2))
forest <- ODRF(Density ~ ., train_data[index, ], split = "mse", parallel = FALSE, ntrees = 50)
prune_forest <- prune(forest, train_data[-index, -1], train_data[-index, 1], useOOB = FALSE)
pred <- predict(prune_forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

`prune.ODT` *pruning of class* `ODT`

Description

Prune `ODT` from bottom to top with validation data based on prediction error.

Usage

```
## S3 method for class 'ODT'
prune(obj, X, y, MaxDepth = 1, ...)
```

Arguments

`obj`	an object of class `ODT`.
`X`	An n by d numeric matrix (preferable) or data frame is used to prune the object of class `ODT`.
`y`	A response vector of length n.
`MaxDepth`	The maximum depth of the tree after pruning. (Default 1)
`...`	Optional parameters to be passed to the low level function.

Details

The leftmost value of the horizontal axis indicates the tree without pruning, while the rightmost value indicates the data without splitting and using the average value as the predicted value.

Value

An object of class ODT and prune.ODT.

- ODT The same result as ODT.
- pruneError Error of validation data after each pruning, misclassification rate (MR) for classification or mean square error (MSE) for regression. The maximum value indicates the tree without pruning, and the minimum value (0) indicates indicates the data without splitting and using the average value as the predicted value.

See Also

ODT plot.prune.ODT prune.ODRF online.ODT

Examples

```
# Classification with Oblique Decision Tree
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
index <- seq(floor(nrow(train_data) / 2))
tree <- ODT(varieties_of_wheat ~ ., train_data[index, ], split = "entropy")
prune_tree <- prune(tree, train_data[-index, -8], train_data[-index, 8])
pred <- predict(prune_tree, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

# Regression with Oblique Decision Tree
data(body_fat)
set.seed(221212)
train <- sample(1:252, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
index <- seq(floor(nrow(train_data) / 2))
tree <- ODT(Density ~ ., train_data[index, ], split = "mse")
prune_tree <- prune(tree, train_data[-index, -1], train_data[-index, 1])
pred <- predict(prune_tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

RandRot *Samples a p x p uniformly random rotation matrix*

Description

Samples a p x p uniformly random rotation matrix via QR decomposition of a matrix with elements sampled iid from a standard normal distribution.

Usage

```
RandRot(p)
```

Arguments

p	The columns of an n by p numeric matrix or data frame.

Value

A p x p uniformly random rotation matrix.

See Also

RotMatPPO RotMatRand RotMatRF RotMatMake

Examples

```
set.seed(220828)
(RandRot(10))
```

RotMatMake *Create rotation matrix used to determine the linear combination of features.*

Description

Create any projection matrix with a self-defined projection matrix function and projection optimization model function

Usage

```
RotMatMake(
  X = NULL,
  y = NULL,
  RotMatFun = "RotMatPPO",
  PPFun = "PPO",
  FunDir = getwd(),
  paramList = NULL,
  ...
)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
y	A response vector of length n.
RotMatFun	A self-defined projection matrix function name, which can also be RotMatRand and RotMatPPO. Note that (, . . .) is necessary.
PPFun	A self-defined projection function name, which can also be PPO. Note that (, . . .) is necessary.
FunDir	The path to the function of the user-defined NodeRotateFun (default current Workspace).
paramList	List of parameters used by the functions RotMatFun and PPFun. If left unchanged, default values will be used, for details see defaults.
. . .	Used to handle superfluous arguments passed in using paramList.

Details

There are two ways for the user to define a projection direction function. The first way is to connect two custom functions with the function RotMatMake(). Specifically, RotMatFun() is defined to determine the variables to be projected, the projection dimensions and the number of projections (the first two columns of the rotation matrix). PPFun() is defined to determine the projection coefficients (the third column of the rotation matrix). After that let the argument RotMatFun="RotMatMake", and the argument paramList must contain the parameters RotMatFun and PPFun. The second way is to define a function directly, and just let the argument RotMatFun be the name of the defined function and let the argument paramList be the arguments list used in the defined function.

Value

A random matrix to use in running ODT.

- Variable: Variables to be projected.
- Number: Number of projections.
- Coefficient: Coefficients of the projection matrix.

See Also

RotMatPPO RotMatRand RotMatRF

Examples

```
set.seed(220828)
X <- matrix(rnorm(1000), 100, 10)
y <- (rnorm(100) > 0) + 0
(RotMat <- RotMatMake(X, y, "RotMatRand", "PPO"))
library(nnet)
(RotMat <- RotMatMake(X, y, "RotMatPPO", "PPO", paramList = list(model = "Log")))

## Define projection matrix function makeRotMat and projection pursuit function makePP.##
##  Note that '...' is necessary.
```

```
makeRotMat <- function(dimX, dimProj, numProj, ...) {
  RotMat <- matrix(1, dimProj * numProj, 3)
  for (np in seq(numProj)) {
    RotMat[(dimProj * (np - 1) + 1):(dimProj * np), 1] <-
      sample(1:dimX, dimProj, replace = FALSE)
    RotMat[(dimProj * (np - 1) + 1):(dimProj * np), 2] <- np
  }
  return(RotMat)
}

makePP <- function(dimProj, prob, ...) {
  pp <- sample(c(1L, -1L), dimProj, replace = TRUE, prob = c(prob, 1 - prob))
  return(pp)
}

RotMat <- RotMatMake(
  RotMatFun = "makeRotMat", PPFun = "makePP",
  paramList = list(dimX = 8, dimProj = 5, numProj = 4, prob = 0.5)
)
head(RotMat)
#>      Variable Number Coefficient
#> [1,]        6      1           1
#> [2,]        8      1           1
#> [3,]        1      1          -1
#> [4,]        4      1          -1
#> [5,]        5      1          -1
#> [6,]        6      2           1


# train ODT with defined projection matrix function
tree <- ODT(X, y,
  split = "entropy", NodeRotateFun = "makeRotMat",
  paramList = list(dimX = ncol(X), dimProj = 5, numProj = 4)
)
# train ODT with defined projection matrix function and projection optimization model function
tree <- ODT(X, y,
  split = "entropy", NodeRotateFun = "RotMatMake", paramList =
    list(
      RotMatFun = "makeRotMat", PPFun = "makePP",
      dimX = ncol(X), dimProj = 5, numProj = 4, prob = 0.5
    )
)
```

RotMatPPO *Create a Projection Matrix: RotMatPPO*

Description

Create a projection matrix using projection pursuit optimization (PPO).

Usage

```
RotMatPPO(
  X,
  y,
  model = "PPR",
  split = "entropy",
  weights = NULL,
  dimProj = min(ceiling(length(y)^0.4), ceiling(ncol(X) * 2/3)),
  numProj = ifelse(dimProj == "Rand", sample(floor(ncol(X)/3), 1),
    ceiling(ncol(X)/dimProj)),
  catLabel = NULL,
  ...
)
```

Arguments

X	An n by d numeric matrix (preferable) or data frame.
y	A response vector of length n.
model	Model for projection pursuit (for details see PPO).
split	One of three criteria, 'gini': gini impurity index (classification), 'entropy': information gain (classification, default) or 'mse': mean square error (regression).
weights	A vector of length same as data that are positive weights. (default NULL)
dimProj	Number of variables to be projected, dimProj=min(ceiling(n^0.4),ceiling(ncol(X)*2/3)) (default) or dimProj="Rand": random from 1 to ncol(X).
numProj	The number of projection directions, when dimProj="Rand" default numProj = sample(ceiling(ncol(X)/3),1) otherwise default numProj=ceiling(ncol(X)/dimProj).
catLabel	A category labels of class list in predictors. (default NULL, for details see Examples of ODT)
...	Used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running ODT.

- Variable: Variables to be projected.
- Number: Number of projections.
- Coefficient: Coefficients of the projection matrix.

See Also

RotMatMake RotMatRand RotMatRF PPO

Examples

```
set.seed(220828)
X <- matrix(rnorm(1000), 100, 10)
y <- (rnorm(100) > 0) + 0
(RotMat <- RotMatPPO(X, y))
(RotMat <- RotMatPPO(X, y, dimProj = "Rand"))
(RotMat <- RotMatPPO(X, y, dimProj = 6, numProj = 4))

# classification
data(seeds)
(PP <- RotMatPPO(seeds[, 1:7], seeds[, 8], model = "Log", split = "entropy"))
(PP <- RotMatPPO(seeds[, 1:7], seeds[, 8], model = "PPR", split = "entropy"))
(PP <- RotMatPPO(seeds[, 1:7], seeds[, 8], model = "LDA", split = "entropy"))

# regression
data(body_fat)
(PP <- RotMatPPO(body_fat[, 2:15], body_fat[, 1], model = "Log", split = "mse"))
(PP <- RotMatPPO(body_fat[, 2:15], body_fat[, 1], model = "Rand", split = "mse"))
(PP <- RotMatPPO(body_fat[, 2:15], body_fat[, 1], model = "PPR", split = "mse"))
```

RotMatRand *Random Rotation Matrix*

Description

Generate rotation matrices by different distributions, and it comes from the library `rerf`.

Usage

```
RotMatRand(
  dimX,
  randDist = "Binary",
  numProj = ceiling(sqrt(dimX)),
  dimProj = "Rand",
  sparsity = ifelse(dimX >= 10, 3/dimX, 1/dimX),
  prob = 0.5,
  lambda = 1,
  catLabel = NULL,
  ...
)
```

Arguments

`dimX`	The number of dimensions.
`randDist`	The probability distribution of the random projection direction, including "Binary": $B\{-1, 1\}$ binomial distribution (default), "Norm":$N(0, 1)$ normal distribution, "Uniform": $U(-1, 1)$ uniform distribution.

numProj	The number of projection directions (default ceiling(sqrt(dimX))).
dimProj	Number of variables to be projected, default dimProj="Rand": random from 1 to dimX.
sparsity	A real number in $(0, 1)$ that specifies the distribution of non-zero elements in the random matrix. When sparsity="pois" means that non-zero elements are generated by the p(lambda) Poisson distribution.
prob	A probability in $(0, 1)$ used for sampling from $-1, 1$ where prob = 0 will only sample -1 and prob = 1 will only sample 1.
lambda	Parameter of the Poisson distribution (default 1).
catLabel	A category labels of class list in predictors. (default NULL, for details see Examples of ODT)
...	Used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running ODT.

- Variable: Variables to be projected.
- Number: Number of projections.
- Coefficient: Coefficients of the projection matrix.

References

Tomita, T. M., Browne, J., Shen, C., Chung, J., Patsolic, J. L., Falk, B., ... & Vogelstein, J. T. (2020). Sparse projection oblique randomer forests. Journal of machine learning research, 21(104).

See Also

RotMatPPO RotMatRF RotMatMake

Examples

```
set.seed(1)
paramList <- list(dimX = 8, numProj = 3, sparsity = 0.25, prob = 0.5)
(RotMat <- do.call(RotMatRand, paramList))
paramList <- list(dimX = 8, numProj = 3, sparsity = "pois")
(RotMat <- do.call(RotMatRand, paramList))
paramList <- list(dimX = 8, randDist = "Norm", dimProj = 5)
(RotMat <- do.call(RotMatRand, paramList))
```

RotMatRF *Create a Projection Matrix: Random Forest (RF)*

Description

Create a projection matrix with coefficient 1 and 0 such that the ODRF (ODT) has the same partition variables as the Random Forest (CART).

Usage

```
RotMatRF(dimX, numProj, catLabel = NULL, ...)
```

Arguments

dimX	The number of dimensions.
numProj	The number of projection directions (default ceiling(sqrt(dimX))).
catLabel	A category labels of class list in predictors. (default NULL, for details see Examples of ODT)
...	Used to handle superfluous arguments passed in using paramList.

Value

A random matrix to use in running ODT.

- Variable: Variables to be projected.
- Number: Number of projections.
- Coefficient: Coefficients of the projection matrix.

See Also

RotMatPPO RotMatRand RotMatMake

Examples

```
paramList <- list(dimX = 8, numProj = 3, catLabel = NULL)
set.seed(2)
(RotMat <- do.call(RotMatRF, paramList))
```

VarImp *Extract variable importance measure*

Description

This is the extractor function for variable importance measures as produced by ODT and ODRF.

Usage

```
VarImp(obj, X = NULL, y = NULL, type = "permutation")
```

Arguments

obj	An object of class ODT and ODRF.
X	An n by d numerical matrix (preferably) or data frame is used in the ODRF.
y	A response vector of length n is used in the ODRF.
type	specifying the type of importance measure. "impurity": mean decrease in node impurity, "permutation" (default): mean decrease in accuracy.

Details

A note from randomForest package, here are the definitions of the variable importance measures.

- The first measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares.
- The second measure is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded. Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees.

Value

A matrix of importance measure, first column is the predictors and second column is Increased error. Misclassification rate (MR) for classification or mean square error (MSE) for regression. The larger the increased error the more important the variable is.

See Also

ODRF Accuracy plot.VarImp

Examples

```
data(body_fat)
y=body_fat[,1]
X=body_fat[,-1]

tree <- ODT(X, y, split = "mse")
```

```
(varimp <- VarImp(tree, type="impurity"))

forest <- ODRF(X, y, split = "mse", parallel = FALSE, ntrees=50)
(varimp <- VarImp(forest, type="impurity"))
(varimp <- VarImp(forest, X, y, type="permutation"))
```

Index