

Package ‘ODRF’

December 31, 2022

Type Package

Title Oblique Decision Random Forest for Classification and Regression

Version 1.0-1

Date 2022-09-21

Author Yu Liu <liuyuchina123@gmail.com>, Yingcun Xia <staxyc@nus.edu.sg>

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. Both can be used for classification and regression as supplements to the classical CART and Random Forest respectively.

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

License GPL (>= 2)

LazyData yes

NeedsCompilation yes

Depends R (>= 3.5.0), partykit, MAVE, mda

Imports Rcpp (>= 0.11.0), stats, doParallel, parallel, foreach, Pursuit, grid, nnet

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.3

Encoding UTF-8

Suggests knitr

VignetteBuilder knitr

R topics documented:

| | |
|-------------------------|----|
| as.party.ODT | 2 |
| best.cut.node | 3 |
| body_fat | 4 |
| breast_cancer | 5 |
| ODRF | 6 |
| ODRF_accuracy | 11 |
| ODT | 12 |
| online | 16 |
| online.ODRF | 17 |
| online.ODT | 18 |

| | |
|------------------------------|-----------|
| plot.ODRF_accuracy | 19 |
| plot.ODT | 20 |
| plot.prune.ODT | 21 |
| plot.VarImp | 22 |
| plot_ODT_depth | 23 |
| PPO | 25 |
| predict.ODRF | 26 |
| predict.ODT | 27 |
| print.ODRF | 29 |
| print.ODT | 29 |
| prune | 30 |
| prune.ODRF | 31 |
| prune.ODT | 32 |
| RotMatMake | 33 |
| RotMatPPO | 35 |
| RotMatRand | 37 |
| RotMatRF | 38 |
| seeds | 39 |
| VarImp | 40 |
| Index | 42 |

| | |
|--------------|---------------------|
| as.party.ODT | ODT <i>as</i> party |
|--------------|---------------------|

Description

To make ODT object to objects of class party.

Usage

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

Arguments

| | |
|--------|---|
| ppTree | An object of class ODT . |
| data | Training data of class data.frame is used to convert the object of class ODRF. and data must be the training data data in ODT . |
| ... | Arguments to be passed to methods |

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software <doi:10.18637/jss.v083.i08>

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 | <i>find best split variable and node.</i> |
|---------------|---|

Description

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

Usage

```
best.cut.node(
  X,
  y,
  type = "i-classification",
  weights = 1,
  MinLeaf = ifelse(type == "regression", 5, 1),
  numLabels = ifelse(type == "regression", 0, length(unique(y)))
)
```

Arguments

| | |
|-----------|--|
| X | An n by d numeric matrix (preferable) or data frame. |
| y | A response vector of length n. |
| type | One of three criteria, 'i-classification': information gain (classification, default), 'g-classification': gini impurity index (classification) or 'regression': mean square error (regression). |
| weights | A vector of values which weigh the samples when considering a split. |
| MinLeaf | The minimum amount of samples in a leaf. |
| numLabels | The number of categories. |

Value

A list which contains:

- BestCutVar: The best split variable.
- BestCutVal: The best split point for the best split variable.
- BestIndex: Each variable corresponds to the min gini impurity index(method='g-classification'), the max information gain(method='i-classification') or the min squared error(method='regression').

Examples

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

body_fat

Body Fat Prediction Dataset

Description

Lists estimates of the percentage of body fat determined by underwater weighing and various body circumference measurements for 252 men. Accurate measurement of body fat is inconvenient/costly and it is desirable to have easy methods of estimating body fat that are not inconvenient/costly.

Format

A data frame with 252 rows and 15 covariate variables and 1 response variable

Details

The variables listed below, from left to right, are:

- Density determined from underwater weighing
- Age (years)
- Weight (lbs)
- Height (inches)
- Neck circumference (cm)
- Chest circumference (cm)
- Abdomen 2 circumference (cm)
- Hip circumference (cm)
- Thigh circumference (cm)
- Knee circumference (cm)
- Ankle circumference (cm)
- Biceps (extended) circumference (cm)
- Forearm circumference (cm)
- Wrist circumference (cm)

Source

<https://www.kaggle.com/datasets/fedesoriano/body-fat-prediction-dataset>

References

Bailey, Covert (1994). Smart Exercise: Burning Fat, Getting Fit, Houghton-Mifflin Co., Boston, pp. 179-186.

See Also

[breast_cancer seeds](#)

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, ])

forest <- ODRF(Density ~ ., train_data, type = "regression", parallel = FALSE)
pred <- predict(forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)

tree <- ODT(Density ~ ., train_data, type = "regression")
pred <- predict(tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

breast_cancer

Breast Cancer Dataset

Description

Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area. The key challenges against it's detection is how to classify tumors into malignant (cancerous) or benign(non cancerous).

Format

A data frame with 569 rows and 30 covariate variables and 1 response variable

Details

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in:

- ID number
- Diagnosis (M = malignant, B = benign)
- Ten real-valued features are computed for each cell nucleus:
 - radius (mean of distances from center to points on the perimeter)
 - texture (standard deviation of gray-scale values)

- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness ($\text{perimeter}^2 / \text{area} - 1.0$)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

Source

<https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset?select=breast-cancer.csv> and [https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(diagnostic\)](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic))

References

Wolberg WH, Street WN, Mangasarian OL. Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Lett.* 1994 Mar 15;77(2-3):163-71.

See Also

[body_fat seeds](#)

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, type = "i-classification", parallel = FALSE)
pred <- predict(forest, test_data[, -1])
# classification error
(mean(pred != test_data[, 1]))

tree <- ODT(diagnosis ~ ., train_data, type = "i-classification")
pred <- predict(tree, test_data[, -1])
# classification error
(mean(pred != test_data[, 1]))
```

Description

Classification and regression implemented by the oblique decision random forest. It is an extension of random forest and include random forest as a special case. It usually produces more accurate predictions, but needs a long computation time.

Usage

```

ODRF(formula, ...)

## S3 method for class 'formula'
ODRF(
  formula,
  data = NULL,
  type = "auto",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  numOOB = 1/3,
  parallel = TRUE,
  numCores = Inf,
  seed = 220924,
  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,
  type = "auto",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  ntrees = 100,
  storeOOB = TRUE,
  replacement = TRUE,
  stratify = TRUE,
  numOOB = 1/3,
  parallel = TRUE,
  numCores = Inf,
  seed = 220924,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 5,
  subset = NULL,

```

```

weights = NULL,
na.action = na.fail,
catLabel = NULL,
Xcat = 0,
Xscale = "Min-max",
TreeRandRotate = FALSE,
...
)

```

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) |
| ... | Optional parameters to be passed to the low level function. |
| data | Training data of class data.frame in which to interpret the variables named in the formula. If data is missing it is obtained from the current environment by formula. |
| type | The criterion used for splitting the nodes. 'i-classification': information gain and 'g-classification': gini impurity index for classification; 'regression': mean square error for regression. 'auto' (default): If the response in data or y is a factor, 'g-classification' is used, otherwise regression is assumed. |
| NodeRotateFun | Name of the function of class character that implements a linear combination of predictors in the split node. including <ul style="list-style-type: none"> "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). |
| numOOB | 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). |
| seed | Random seeds in order to reproduce results. |
| 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 1 for classification, 5 for regression. |

| | |
|----------------|--|
| 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. |
| TreeRandRotate | If or not to randomly rotate the Training data before building the tree (default FALSE). |
| X | An n by d numeric matrix (preferable) or data frame. |
| 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.
- 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.
- type, 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.

Author(s)

Yu Liu and Yingcun Xia

References

Zhan H, Liu Y, Xia Y. Consistency of The Oblique Decision Tree and Its Random Forest[J]. arXiv preprint arXiv:2211.12653, 2022.

Tomita T M, Browne J, Shen C, et al. Sparse projection oblique randomer forests[J]. Journal of machine learning research, 2020, 21(104).

See Also

[online.ODRF](#) [prune.ODRF](#) [predict.ODRF](#) [print.ODRF](#) [ODRF_accuracy](#) [VarImp](#)

Examples

```
# Classification with Oblique Decision Random Forest.
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., train_data, type = "i-classification",
  parallel = FALSE)
pred <- predict(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, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
forest <- ODRF(Density ~ ., train_data, type = "regression", parallel = FALSE)
pred <- predict(forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)

### Train ODRF on one-of-K encoded categorical data ###
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, type = "i-classification", 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 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"

forest <- ODRF(X, y, type = "g-classification", Xcat = c(1, 2),
  catLabel = catLabel, parallel = FALSE)

```

ODRF_accuracy

accuracy of oblique decision random forest

Description

Prediction accuracy of ODRF at different tree sizes.

Usage

```
ODRF_accuracy(ppForest, data, newdata = NULL)
```

Arguments

| | |
|----------|---|
| ppForest | 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, let it be data. |

Value

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

See Also

[ODRF](#) [plot.ODRF](#) [accuracy](#)

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, type = "i-classification", parallel = FALSE)
(error <- ODRF_accuracy(forest, train_data, test_data))
```

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(formula, ...)

## S3 method for class 'formula'
ODT(
  formula,
  data = NULL,
  type = "auto",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 5,
  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,
  type = "auto",
  NodeRotateFun = "RotMatPPO",
  FunDir = getwd(),
  paramList = NULL,
  MaxDepth = Inf,
  numNode = Inf,
  MinLeaf = 5,
  Levels = NULL,
  subset = NULL,
  weights = NULL,
  na.action = na.fail,
  catLabel = NULL,
  Xcat = 0,
  Xscale = "Min-max",
  TreeRandRotate = FALSE,
  ...
)

```

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) |
| ... | Optional parameters to be passed to the low level function. |
| data | Training data of class data.frame in which to interpret the variables named in the formula. If data is missing it is obtained from the current environment by formula. |
| type | The criterion used for splitting the nodes. 'i-classification': information gain and 'g-classification': gini impurity index for classification; 'regression': mean square error for regression; 'auto' (default): If the response in data or y is a factor, 'g-classification' is used, otherwise regression is assumed. |
| NodeRotateFun | Name of the function of class character that implements a linear combination of predictors in the split node. including <ul style="list-style-type: none"> "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 . |
| 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 1 for classification, 5 for regression. |
| Levels | The category label of the response variable when type is not equal to 'regression'. |
| 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. |
| TreeRandRotate | If or not to randomly rotate the Training data before building the tree (default FALSE). |
| X | An n by d numeric matrix (preferable) or data frame. |
| y | A response vector of length n. |

Value

An object of class ODT Containing a list 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.
- projections: Projection direction for each node.
- 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 type.
 - childNode: Record the number of child nodes after each splitting.
 - nodeDepth: Record the depth of the tree where each node is located.
- type, 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 tree.
- tree: The list of tree related parameters used to build the tree.

Author(s)

Yu Liu and Yingcun Xia

References

Zhan H, Liu Y, Xia Y. Consistency of The Oblique Decision Tree and Its Random Forest[J]. arXiv preprint arXiv:2211.12653, 2022.

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, type = "i-classification")
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, type = "regression")
pred <- predict(tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

```
### Train ODT on one-of-K encoded categorical data ###
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(y ~ X, type = "i-classification", 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, type = "g-classification", Xcat = c(1, 2), catLabel = catLabel)

```

online

online structure learning for class ODT and ODRF.

Description

[ODT](#) and [ODRF](#) are constantly updated by multiple batches of data to optimize the model. [online](#) is a S3 method for class ODT and ODRF.

Usage

```
online(obj, ...)
```

Arguments

obj an object of class ODT or ODRF.

... For other parameters related to class obj, see ODT or ODRF.

Value

object of class ODT or ODRF.

See Also

[ODT](#) [ODRF](#) [online.ODT](#) [online.ODRF](#)

online.ODRF

using training data to update an existing ODRF.

Description

Update existing [ODRF](#) using batches of data to improve the model.

Usage

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

Arguments

| | |
|----------|---|
| ppForest | An object of class ODRF. |
| X | An n by d numeric matrix (preferable) or data frame is used to update the object of class ODRF. |
| y | A response vector of length n is used to update the object of class ODRF. |
| weights | Vector of non-negative observational weights; fractional weights are allowed (default NULL). |
| ... | 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, 100)
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, ],
  type = "i-classification", parallel = FALSE)
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, 100)
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, ],
  type = "regression", parallel = FALSE)
online.forest <- online(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 | <i>using training data to update an existing ODT.</i> |
|------------|---|

Description

Update existing [ODT](#) using batches of data to improve the model.

Usage

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

Arguments

| | |
|---------|--|
| ppTree | an object of class ODT. |
| X | An n by d numeric matrix (preferable) or data frame is used to update the object of class ODT. |
| y | A response vector of length n is used to update the object of class ODT. |
| weights | Vector of non-negative observational weights; fractional weights are allowed (default NULL). |
| ... | 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, ],
  type = "i-classification")
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, ], type = "regression")
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.ODRF_accuracy *plot method for ODRF_accuracy objects*

Description

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

Usage

```
## S3 method for class 'ODRF_accuracy'
plot(
  Err,
  lty = 1,
  digits = NULL,
  main = paste0("Oblique ", ifelse(Err$type == "regression", "Regression",
    "Classification"), " Forest"),
  ...
)
```

Arguments

Err Object of class [ODRF_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 ODRF_accuracy](#)

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, type = "i-classification", parallel = FALSE)
(error <- ODRF_accuracy(forest, train_data, test_data))
plot(error)
```

| | |
|----------|---|
| plot.ODT | <i>to plot an oblique decision tree</i> |
|----------|---|

Description

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

Usage

```
## S3 method for class 'ODT'
plot(
  ppTree,
  font.size = 17,
  width.size = 1,
  xadj = 0,
  main = paste0("Oblique ", ifelse(ppTree$type == "regression", "Regression",
    "Classification"), " Tree"),
  sub = NULL,
  ...
)
```

Arguments

| | |
|------------|--|
| ppTree | 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. |

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software <doi:10.18637/jss.v083.i08>

See Also

[ODT as.party plot_ODT_depth](#)

Examples

```
data(iris)
tree <- ODT(Species ~ ., data = iris, type = "i-classification")
plot(tree)
```

| | |
|----------------|---|
| plot.prune.ODT | <i>to plot pruned oblique decision tree</i> |
|----------------|---|

Description

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

Usage

```
## S3 method for class 'prune.ODT'
plot(
  ppTree,
  position = "topleft",
  digits = NULL,
  main = paste0("Oblique ", ifelse(ppTree$type == "regression", "Regression",
    "Classification"), " Tree"),
  ...
)
```

Arguments

| | |
|----------|--|
| ppTree | An object of class <code>prune.ODT</code> . |
| position | Position of the curve label. |
| 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

Error of validation data after each pruning, misclassification rate (MR) for classification or mean square error (MSE) for regression.

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, type = "regression")
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 a Oblique Decision Random Forest.

Usage

```
## S3 method for class 'VarImp'
plot(
  varImp,
  nvar = min(30, nrow(varImp$varImp)),
  digits = NULL,
  main = paste0("Oblique ", ifelse(varImp$type == "regression", "Regression",
    "Classification"), " Forest"),
  ...
)
```

Arguments

| | |
|--------|--|
| varImp | An object of class VarImp . |
| nvar | How many 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

A matrix of importance measure, first column for each predictor variable and second column is Increased error. Error is misclassification rate (MR) for classification or mean square error (MSE) for regression.

See Also

[ODRF VarImp](#)

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data,
  type = "i-classification", parallel = FALSE)
(varimp <- VarImp(forest, train_data[, -1], train_data[, 1]))
plot(varimp, digits = 0)
```

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,
  type = "i-classification",
  NodeRotateFun = "RotMatPPO",
  paramList = NULL,
  digits = NULL,
  main = paste0("Oblique ", ifelse(type == "regression", "Regression", "Classification"),
    " Tree"),
  ...
)
```

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 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, let it be data. |
| type | The criterion used for splitting the nodes. 'i-classification': information gain and 'g-classification': gini impurity index for classification; 'regression': mean square error for regression. 'auto' (default): If the response in data is a factor, 'g-classification' is used, otherwise regression is assumed. |
| NodeRotateFun | Name of the function of class character that implements a linear combination of predictors in the split node. including <ul style="list-style-type: none"> "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, type = "regression")
```


PPO

*Projection Pursuit Optimization***Description**

Find the optimal projection using various projectin pursuit models.

Usage

```
PPO(X, y, model = "PPR", type = "i-classification", weights = NULL, ...)
```

Arguments

| | |
|----------------------|---|
| <code>X</code> | An n by d numeric matrix (preferable) or data frame. |
| <code>y</code> | A response vector of length n. |
| <code>model</code> | Model for projection pursuit. <ul style="list-style-type: none"> • "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 type must be 'i-classification' or 'g-classification'. • "LDA", "PDA", "Lr", "GINI", and "ENTROPY" from library PPtreeViz. • The following models based on Pursuit. <ul style="list-style-type: none"> – "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 |
| <code>type</code> | The criterion used for splitting the variable. 'g-classification': gini impurity index (classification, default), 'i-classification': information gain (classification) or 'regression': mean square error (regression). |
| <code>weights</code> | Vector of non-negative observational weights; fractional weights are allowed (default NULL). |
| <code>...</code> | 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", type = "i-classification"))
(PP <- PPO(seeds[, 1:7], seeds[, 8], model = "PPR", type = "i-classification"))
(PP <- PPO(seeds[, 1:7], seeds[, 8], model = "LDA", type = "i-classification"))

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

predict.ODRF

predict based on ODRF objects

Description

Prediction of ODRF for an input matrix or data frame.

Usage

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

Arguments

- | | |
|-------------|---|
| ppForest | An object of class ODRF, as that 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. |
| 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) |

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 ppForest\$type is regression, a vector of tree weights is returned.
- tree: it is a matrix where each column contains prediction by a tree in the forest.

References

Zhan H, Liu Y, Xia Y. Consistency of The Oblique Decision Tree and Its Random Forest[J]. arXiv preprint arXiv:2211.12653, 2022.

See Also

[ODRF predict.ODT](#)

Examples

```
# Classification with Oblique Decision Random Forest
data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., train_data,
  type = "i-classification", parallel = FALSE)
pred <- predict(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, 100)
train_data <- data.frame(body_fat[train, ])
test_data <- data.frame(body_fat[-train, ])
forest <- ODRF(Density ~ ., train_data, type = "regression", parallel = FALSE)
pred <- predict(forest, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

predict.ODT

predict method for ODT objects

Description

Prediction of ODT for an input matrix or data frame.

Usage

```
## S3 method for class 'ODT'
predict(ppTree, Xnew, leafnode = FALSE)
```

Arguments

| | |
|-----------------------|--|
| <code>ppTree</code> | An object of class ODT, as that created by the function ODT . |
| <code>Xnew</code> | An n by d numeric matrix (preferable) or data frame. The rows correspond to observations and columns correspond to features. |
| <code>leafnode</code> | If or not output the leaf node sequence number that <code>Xnew</code> is partitioned. (default FALSE) |

Value

A vector of the following:

- prediction: the predicted response of the new data.
- leafnode: the leaf node sequence number that the new data is partitioned.

References

Zhan H, Liu Y, Xia Y. Consistency of The Oblique Decision Tree and Its Random Forest[J]. arXiv preprint arXiv:2211.12653, 2022.

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, type = "i-classification")
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, type = "regression")
pred <- predict(tree, test_data[, -1])
# estimation error
mean((pred - test_data[, 1])^2)
```

| | |
|------------|-------------------|
| print.ODRF | <i>print ODRF</i> |
|------------|-------------------|

Description

Print contents of ODRF object.

Usage

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

Arguments

| | |
|----------|---|
| ppForest | An object of class ODRF . |
| ... | Arguments to be passed to methods. |

See Also

[ODRF](#)

Examples

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

| | |
|-----------|-------------------------|
| print.ODT | <i>print ODT result</i> |
|-----------|-------------------------|

Description

Print the oblique decision tree structure.

Usage

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

Arguments

| | |
|------------|---|
| ppTree | 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. |

References

Lee, EK(2017) PPtreeViz: An R Package for Visualizing Projection Pursuit Classification Trees, Journal of Statistical Software <doi:10.18637/jss.v083.i08>

See Also

[ODT](#)

Examples

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

| | |
|-------|--------------------------|
| prune | <i>prune</i> ODT or ODRF |
|-------|--------------------------|

Description

Prune ODT or ODRF from bottom to top with validation data based on prediction error, and prune is a S3 method for class ODT and ODRF.

Usage

```
prune(obj, ...)
```

Arguments

| | |
|-----|--|
| obj | An object of class ODT or ODRF. |
| ... | For other parameters related to class obj, see ODT or ODRF . |

Value

An object of class ODT and `prune.ODT`.

See Also

[ODT](#) [ODRF](#) [prune.ODT](#) [prune.ODRF](#)

| | |
|------------|-------------------------------|
| prune.ODRF | <i>Pruning of class ODRF.</i> |
|------------|-------------------------------|

Description

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

Usage

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

Arguments

| | |
|----------|--|
| ppForest | 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, 100)
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, ],
  type = "i-classification", parallel = FALSE)
prune.forest <- prune(forest, train_data[-index, -8], train_data[-index, 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, 100)
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, ], type = "regression", parallel = FALSE)
prune.forest <- prune(forest, train_data[-index, -1], train_data[-index, 1])
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(ppTree, X, y, MaxDepth = 1, ...)

```

Arguments

| | |
|----------|---|
| ppTree | 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. |

Value

An object of class ODT and prune.ODT.

ppTree 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.

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, ], type = "i-classification")
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, ], type = "regression")
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)
```

RotMatMake

Create rotation matrix used to determine 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 <code>(, ...)</code> is necessary. |
| PPFun | A self-defined projection matrix function, which can also be PPO . Note that <code>(, ...)</code> 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 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 used in the defined function; the second way is to use the function RotMatMake following the way in the example below. Note that the name of the defined function cannot be the name of an existing function in the ODRF package

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 and projection optimization model function.##
## 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, type = "i-classification", 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, type = "i-classification", 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",
  type = "i-classification",
  weights = NULL,
  dimProj = min(ceiling(length(y)^0.4), ceiling(ncol(X) * 2/3)),
  numProj = ifelse(dimProj == "Rand", max(5, sample(floor(ncol(X)/3), 1)), max(5,
    ceiling(ncol(X)/dimProj))),
  catLabel = NULL,
  ...
)

```

Arguments

| | |
|--------------------|--|
| <code>X</code> | An n by d numeric matrix (preferable) or data frame. |
| <code>y</code> | A response vector of length n . |
| <code>model</code> | Model for projection pursuit (for details see PPO). |

| | |
|----------|--|
| type | The criterion used for splitting the variable. 'i-classification': information gain (classification, default), 'g-classification': gini impurity index (classification) or 'regression': 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(p0/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", type = "i-classification"))
(PP <- RotMatPPO(seeds[, 1:7], seeds[, 8], model = "PPR", type = "i-classification"))
(PP <- RotMatPPO(seeds[, 1:7], seeds[, 8], model = "LDA", type = "i-classification"))

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

RotMatRand

*Random Rotation Matrix***Description**

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

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

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

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, et al. Sparse projection oblique randomer forests[J]. Journal of machine learning research, 2020, 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 such that the ODRF (ODT) has the same partition variables as the random forest (tree).

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))
```

seeds

seeds Data Set

Description

Measurements of geometrical properties of kernels belonging to three different varieties of wheat. A soft X-ray technique and GRAINS package were used to construct all seven, real-valued attributes.

Format

A data frame with 209 rows and 7 covariate variables and 1 response variable.

Details

The variables listed below, from left to right, are:

- area A
- perimeter P
- compactness $C = 4 \cdot \pi \cdot A / P^2$
- length of kernel
- width of kernel
- asymmetry coefficient
- length of kernel groove
- varieties of wheat (1, 2, 3 for Kama, Rosa and Canadian respectively)

Source

<https://archive.ics.uci.edu/ml/datasets/seeds>

References

M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, S. Zak, 'A Complete Gradient Clustering Algorithm for Features Analysis of X-ray Images', in: Information Technologies in Biomedicine, Ewa Pietka, Jacek Kawa (eds.), Springer-Verlag, Berlin-Heidelberg, 2010, pp. 15-24.

See Also

[body_fat breast_cancer](#)

Examples

```

data(seeds)
set.seed(221212)
train <- sample(1:209, 100)
train_data <- data.frame(seeds[train, ])
test_data <- data.frame(seeds[-train, ])

forest <- ODRF(varieties_of_wheat ~ ., train_data,
  type = "i-classification", parallel = FALSE
)
pred <- predict(forest, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

tree <- ODT(varieties_of_wheat ~ ., train_data, type = "i-classification")
pred <- predict(tree, test_data[, -8])
# classification error
(mean(pred != test_data[, 8]))

```

VarImp

*variable importance of oblique decision random forest***Description**

Variable importance is computed from permuting OOB data.

Usage

```
VarImp(ppForest, X, y)
```

Arguments

| | |
|----------|--|
| ppForest | An object of class 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. |

Details

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

See Also

[ODRF plot.VarImp](#)

Examples

```
data(breast_cancer)
set.seed(221212)
train <- sample(1:569, 200)
train_data <- data.frame(breast_cancer[train, -1])
test_data <- data.frame(breast_cancer[-train, -1])

forest <- ODRF(diagnosis ~ ., train_data, type = "i-classification", parallel = FALSE)
(varimp <- VarImp(forest, train_data[, -1], train_data[, 1]))
```

Index

- * **CART**
 - ODRF, 6
 - ODT, 12
- * **Forest**
 - RotMatRF, 38
- * **Random**
 - RotMatRand, 37
 - RotMatRF, 38
- * **Rotation**
 - RotMatRand, 37
- * **datasets**
 - body_fat, 4
 - breast_cancer, 5
 - seeds, 39
- * **decision**
 - ODRF, 6
 - ODT, 12
- * **forest**
 - ODRF, 6
 - ODT, 12
 - plot.ODRF_accuracy, 19
 - print.ODRF, 29
- * **oblique**
 - ODRF, 6
 - ODT, 12
- * **projection**
 - ODRF, 6
 - ODT, 12
 - PPO, 25
 - RotMatPPO, 35
- * **pursuit**
 - ODRF, 6
 - ODT, 12
 - PPO, 25
 - RotMatPPO, 35
- * **random**
 - ODRF, 6
 - ODT, 12
- * **tree**
 - as.party.ODT, 2
 - ODRF, 6
 - ODT, 12
 - plot.ODT, 20
 - plot.prune.ODT, 21
 - plot_ODT_depth, 23
 - print.ODT, 29
- as.party, 15, 21
- as.party.ODT, 2
- best.cut.node, 3
- body_fat, 4, 6, 39
- breast_cancer, 5, 5, 39
- defaults, 8, 13, 24, 34
- model.frame, 8, 13, 24
- nnet, 25
- ODRF, 6, 11, 12, 16, 17, 20, 23, 26, 27, 29–31, 40
- ODRF_accuracy, 10, 11, 19, 20
- ODT, 2, 9, 12, 16–18, 21, 22, 24, 28–30, 32, 34, 36–38
- online, 16
- online.ODRF, 10, 17, 17, 18, 31
- online.ODT, 15, 17, 18, 32
- par, 19
- party, 2
- plot.ODRF_accuracy, 12, 19
- plot.ODT, 15, 20, 24
- plot.prune.ODT, 21, 32
- plot.VarImp, 22, 40
- plot_ODT_depth, 15, 21, 23
- PPO, 8, 13, 24, 25, 34–36
- ppr, 25
- predict.ODRF, 10, 26, 28
- predict.ODT, 15, 27, 27
- print.ODRF, 10, 29
- print.ODT, 15, 29
- prune, 30
- prune.ODRF, 10, 17, 30, 31, 32
- prune.ODT, 15, 18, 22, 30, 31, 32
- Pursuit, 25
- RotMatMake, 8, 13, 24, 33, 36, 38

RotMatPPO, [8](#), [13](#), [24](#), [26](#), [34](#), [35](#), [38](#)

RotMatRand, [8](#), [13](#), [24](#), [34](#), [36](#), [37](#), [38](#)

RotMatRF, [8](#), [13](#), [24](#), [34](#), [36](#), [38](#), [38](#)

seeds, [5](#), [6](#), [39](#)

terms.object, [9](#), [14](#)

VarImp, [10](#), [23](#), [40](#)