Package ‘dipm’

October 27, 2022

Title Depth Importance in Precision Medicine (DIPM) Method

Version 1.9

Date 2022-10-27

Maintainer Cai Li <cai.li.stats@gmail.com>

Description An implementation by Chen, Li, and Zhang (2022) <doi:10.1093/bioadv/vbac041> of the Depth Importance in Precision Medicine (DIPM) method in Chen and Zhang (2022) <doi:10.1093/biostatistics/kxaa021> and Chen and Zhang (2020) <doi:10.1007/978-3-030-46161-4_16>. The DIPM method is a classification tree that searches for subgroups with especially poor or strong performance in a given treatment group.

Depends R (>= 3.0.0)

Imports stats, utils, survival, partykit (>= 1.2-6), ggplot2, grid

NeedsCompilation yes

License GPL (>= 2)

Encoding UTF-8

LazyLoad yes

Repository CRAN

RoxygenNote 7.1.1

Author Cai Li [aut, cre],
Victoria Chen [aut],
Heping Zhang [aut]

Date/Publication 2022-10-27 15:05:15 UTC

R topics documented:

dipm *Depth Importance in Precision Medicine (DIPM)*

Description

This function creates a classification tree designed to identify subgroups in which subjects perform especially well or especially poorly in a given treatment group.

Usage

```
dipm(
  formula,
  data,
  types = NULL,
  nmin = 5,
  nmin2 = 5,
  ntree = NULL,
  mtry = Inf,
  maxdepth = Inf,
  maxdepth2 = Inf,
  print = TRUE,
  dataframe = FALSE,
  prune = FALSE
)
```

Arguments

`formula`	A description of the model to be fit with format `Y ~ treatment	X1 + X2` for data with a continuous outcome variable Y and `Surv(Y, delta) ~ treatment	X1 + X2` for data with a right-censored survival outcome variable Y and a status indicator delta
`data`	A matrix or data frame of the data
`types`	A vector, data frame, or matrix of the types of each variable in the data; if left blank, the default is to assume all of the candidate split variables are ordinal; otherwise, all variables in the data must be specified, and the possible variable types are: "response", "treatment", "status", "binary", "ordinal", and "nominal" for outcome variable Y, the treatment variable, the status indicator (if applicable), binary candidate split variables, ordinal candidate split variables, and nominal candidate split variables respectively
`nmin`	An integer specifying the minimum node size of the overall classification tree
`nmin2`	An integer specifying the minimum node size of embedded trees
`ntree`	An integer specifying the number of embedded trees to construct at each node of the overall classification tree; if left blank, the default value of `ceiling(min(max(sqrt(n), sqrt(nc)), 1000))` will be used if `mtry = Inf` below and `ceiling(min(max(n, nc), 1000))` otherwise; `n` is the total sample size of the data, and `nc` is the total number of candidate split variables

`mtry`	An integer specifying the number of candidate split variables to randomly select at each node of embedded trees; if `mtry` is set equal to the default value of Inf, then all possible splits of all candidate split variables are considered at the nodes of the embedded trees; otherwise, a recommended value of `mtry` to use is the square root of the total number of candidate split variables rounded up to the nearest integer
`maxdepth`	An integer specifying the maximum depth of the overall classification tree; this argument is optional but useful for shortening computation time; if left blank, the default is to grow the full tree until the minimum node size `nmin` is reached
`maxdepth2`	An integer specifying the maximum depth of embedded trees; this argument is optional but useful for shortening computation time; if left blank, the default is to grow each full, embedded tree until the minimum node size `nmin2` is reached
`print`	A boolean (TRUE/FALSE) value, where TRUE prints a more readable version of the final tree to the screen
`dataframe`	A boolean (TRUE/FALSE) value, where TRUE returns the final tree as a dataframe
`prune`	A boolean (TRUE/FALSE) value, where TRUE prunes the final tree using `pmprune` function

Details

This function creates a classification tree to identify subgroups relevant to the precision medicine setting. At each node of the classification tree, a random forest of so-called embedded trees are fit and used to calculate a depth variable importance score for each candidate split variable in the data. The candidate split variable with the largest variable importance score is identified as the best split variable of the node. Then, all possible splits of the selected split variable are considered, and the split with the greatest split criteria value is finally selected as the best split of the best variable.

The depth variable importance score was originally proposed by Chen et al. (2007), and the score has been adapted to the precision medicine setting here. The depth variable importance score is a relatively simple measure that takes into account two components: the depth of a split in a tree and the strength of the split. The strength of the split is captured with a G test statistic that may be modified depending on the type of analysis at hand. When the outcome variable is continuous, G is the test statistic that tests the significance of the split by treatment interaction term in a linear regression model. When the outcome variable is a right-censored survival time, G is the test statistic that tests the significance of the split by interaction term in a Cox proportional hazards model.

When using `dipm`, note the following requirements for the supplied data. First, the dataset must contain an outcome variable Y and a treatment variable. If Y is a right-censored survival time outcome, then there must also be a status indicator delta, where values of 1 denote the occurrence of the (harmful) event of interest, and values of 0 denote censoring. If there are only two treatment groups, then the two possible values must be 0 or 1. If there are more than two treatment groups, then the possible values must be integers starting from 1 to the total number of treatment assignments. In regard to the candidate split variables, if a variable is binary, then the variable must take values of 0 or 1. If a variable is nominal, then the values must be integers starting from 1 to the total number of categories. There cannot be any missing values in the dataset. For candidate split variables with missing values, the missings together (MT) method proposed by Zhang et al. (1996) is helpful.

Value

`dipm` returns the final classification tree as a `party` object by default or a data frame. See Hothorn and Zeileis (2015) for details. The data frame contains the following columns of information:

`node`	Unique integer values that identify each node in the tree, where all of the nodes are indexed starting from 1
`splitvar`	Integers that represent the candidate split variable used to split each node, where all of the variables are indexed starting from 1; for terminal nodes, i.e., nodes without child nodes, the value is set equal to NA
`splitvar_name`	The names of the candidate split variables used to split each node obtained from the column names of the supplied data; for terminal nodes, the value is set equal to NA
`type`	Characters that denote the type of each candidate split variable; "bin" is for binary variables, "ord" for ordinal, and "nom" for nominal; for terminal nodes, the value is set equal to NA
`splitval`	Values of the left child node of the current split/node; for binary variables, a value of 0 is printed, and subjects with values of 0 for the current `splitvar` are in the left child node, while subjects with values of 1 are in the right child node; for ordinal variables, `splitval` is numeric and implies that subjects with values of the current `splitvar` less than or equal to `splitval` are in the left child node, while the remaining subjects with values greater than `splitval` are in the right child node; for nominal variables, the `splitval` is a set of integers separated by commas, and subjects in that set of categories are in the left child node, while the remaining subjects are in the right child node; for terminal nodes, the value is set equal to NA
`lchild`	Integers that represent the index (i.e., `node` value) of each node's left child node; for terminal nodes, the value is set equal to NA
`rchild`	Integers that represent the index (i.e., `node` value) of each node's right child node; for terminal nodes, the value is set equal to NA
`depth`	Integers that specify the depth of each node; the root node has depth 1, its children have depth 2, etc.
`nsubj`	Integers that count the total number of subjects within each node
`besttrt`	Integers that denote the identified best treatment assignment of each node

References

Chen, V., Li, C., and Zhang, H. (2022). dipm: an R package implementing the Depth Importance in Precision Medicine (DIPM) tree and Forest-based method. *Bioinformatics Advances*, **2**(1), vbac041.

Chen, V. and Zhang, H. (2022). Depth importance in precision medicine (DIPM): A tree-and forest-based method for right-censored survival outcomes. *Biostatistics* **23**(1), 157-172.

Chen, V. and Zhang, H. (2020). Depth importance in precision medicine (DIPM): a tree and forest based method. In *Contemporary Experimental Design, Multivariate Analysis and Data Mining*, 243-259.

Chen, X., Liu, C.-T., Zhang, M., and Zhang, H. (2007). A forest-based approach to identifying gene and gene-gene interactions. *Proceedings of the National Academy of Sciences of the United States of America* **204**, 19199-19203.

Zhang, H., Holford, T., and Bracken, M.B. (1996). A tree-based method of analysis for prospective studies. *Statistics in Medicine* **15**, 37-49.

Hothorn, T. and Zeileis, A. (2015). partykit: a modular toolkit for recursive partytioning in R. *The Journal of Machine Learning Research* **16**(1), 3905-3909.

See Also

spmtree

Examples

```
#
# ... an example with a continuous outcome variable
#     and two treatment groups
#

N = 100
set.seed(123)

# generate binary treatments
treatment = rbinom(N, 1, 0.5)

# generate candidate split variables
X1 = rnorm(n = N, mean = 0, sd = 1)
X2 = rnorm(n = N, mean = 0, sd = 1)
X3 = rnorm(n = N, mean = 0, sd = 1)
X4 = rnorm(n = N, mean = 0, sd = 1)
X5 = rnorm(n = N, mean = 0, sd = 1)
X = cbind(X1, X2, X3, X4, X5)
colnames(X) = paste0("X", 1:5)

# generate continuous outcome variable
calculateLink = function(X, treatment){

    ((X[, 1] <= 0) & (X[, 2] <= 0)) *
        (25 * (1 - treatment) + 8 * treatment) +

    ((X[, 1] <= 0) & (X[, 2] > 0)) *
        (18 * (1 - treatment) + 20 * treatment) +

    ((X[, 1] > 0) & (X[, 3] <= 0)) *
        (20 * (1 - treatment) + 18 * treatment) +

    ((X[, 1] > 0) & (X[, 3] > 0)) *
        (8 * (1 - treatment) + 25 * treatment)
}
```

```
Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# fit a dipm classification tree
tree1 = dipm(Y ~ treatment | ., data, mtry = 1, maxdepth = 3)
# predict optimal treatment for new subjects
predict(tree1, newdata = head(data),
FUN = function(n)  as.numeric(n$info$opt_trt))


#
# ... an example with a continuous outcome variable
#     and three treatment groups
#

N = 600
set.seed(123)

# generate treatments
treatment = sample(1:3, N, replace = TRUE)

# generate candidate split variables
X1 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X2 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X3 = sample(1:4, N, replace = TRUE)
X4 = sample(1:5, N, replace = TRUE)
X5 = rbinom(N, 1, 0.5)
X6 = rbinom(N, 1, 0.5)
X7 = rbinom(N, 1, 0.5)
X = cbind(X1, X2, X3, X4, X5, X6, X7)
colnames(X) = paste0("X", 1:7)

# generate continuous outcome variable
calculateLink = function(X,treatment){

    10.2 - 0.3 * (treatment == 1) - 0.1 * X[, 1] +
    2.1 * (treatment == 1) * X[, 1] +
    1.2 * X[, 2]
}

Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# create vector of variable types
types = c(rep("ordinal", 2), rep("nominal", 2), rep("binary", 3),
        "response", "treatment")
```

```
# fit a dipm classification tree
tree2 = dipm(Y ~ treatment | ., data, types = types, maxdepth = 2)

#
# ... an example with a survival outcome variable
#     and two treatment groups
#

N = 300
set.seed(321)

# generate binary treatments
treatment = rbinom(N, 1, 0.5)

# generate candidate split variables
X1 = rnorm(n = N, mean = 0, sd = 1)
X2 = rnorm(n = N, mean = 0, sd = 1)
X3 = rnorm(n = N, mean = 0, sd = 1)
X4 = rnorm(n = N, mean = 0, sd = 1)
X5 = rnorm(n = N, mean = 0, sd = 1)
X = cbind(X1, X2, X3, X4, X5)
colnames(X) = paste0("X", 1:5)

# generate survival outcome variable
calculateLink = function(X, treatment){

    X[, 1] + 0.5 * X[, 3] + (3 * treatment - 1.5) * (abs(X[, 5]) - 0.67)
}

Link = calculateLink(X, treatment)
T = rexp(N, exp(-Link))
C0 = rexp(N, 0.1 * exp(X[, 5] + X[, 2]))
Y = pmin(T, C0)
delta = (T <= C0)

# combine variables in a data frame
data = data.frame(X, Y, delta, treatment)

# fit a dipm classification tree
tree3 = dipm(Surv(Y, delta) ~ treatment | .,data, ntree = 1, maxdepth = 2,
           maxdepth2 = 6)

#
# ... an example with a survival outcome variable
#     and four treatment groups
#

N = 800
set.seed(321)

# generate treatments
treatment = sample(1:4, N, replace = TRUE)
```

```
# generate candidate split variables
X1 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X2 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X3 = sample(1:4, N, replace = TRUE)
X4 = sample(1:5, N, replace = TRUE)
X5 = rbinom(N, 1, 0.5)
X6 = rbinom(N, 1, 0.5)
X7 = rbinom(N, 1, 0.5)
X = cbind(X1, X2, X3, X4, X5, X6, X7)
colnames(X) = paste0("X", 1:7)

# generate survival outcome variable
calculateLink = function(X, treatment, noise){

    -0.2 * (treatment == 1) +
    -1.1 * X[, 1] +
    1.2 * (treatment == 1) * X[, 1] +
    1.2 * X[, 2]
}

Link = calculateLink(X, treatment)
T = rweibull(N, shape=2, scale = exp(Link))
Cnoise = runif(n = N) + runif(n = N)
C0 = rexp(N, exp(0.3 * -Cnoise))
Y = pmin(T, C0)
delta = (T <= C0)

# combine variables in a data frame
data = data.frame(X, Y, delta, treatment)

# create vector of variable types
types = c(rep("ordinal", 2), rep("nominal", 2), rep("binary", 3),
        "response", "status", "treatment")

# fit two dipm classification trees
tree4 = dipm(Surv(Y, delta) ~ treatment | ., data, types = types, ntree = 1,
           maxdepth = 2, maxdepth2 = 6)
tree5 = dipm(Surv(Y, delta) ~ treatment | X3 + X4, data, types = types, ntree = 1,
           maxdepth = 2, maxdepth2 = 6)
```

node_dipm *Panel-Generator for Visualization of A Precision Medicine Tree*

Description

This function provides a new plot method for `dipm` and `spmtree`. It visualizes stratified treatment groups through boxplots for a continuous outcome and survival plots for a survival outcome, respectively.

Usage

```
node_dipm(obj, ...)
```

Arguments

`obj`	A `party` tree object returned from either the `dipm()` or `spmtree()` function
`...`	Arguments passed on to plotfun

Details

This function visualizes the precision medicine trees proposed in Chen and Zhang (2020a, b).

Value

No return value, called for plot

References

Chen, V., Li, C., and Zhang, H. (2022). dipm: an R package implementing the Depth Importance in Precision Medicine (DIPM) tree and Forest-based method. *Bioinformatics Advances*, **2**(1), vbac041.

Chen, V. and Zhang, H. (2020). Depth importance in precision medicine (DIPM): a tree and forest based method. In *Contemporary Experimental Design, Multivariate Analysis and Data Mining*, 243-259.

Chen, V. and Zhang, H. (2022). Depth importance in precision medicine (DIPM): A tree-and forest-based method for right-censored survival outcomes. *Biostatistics* **23**(1), 157-172.

Seibold, H., Zeileis, A., and Hothorn, T. (2019). model4you: An R package for personalised treatment effect estimation. *Journal of Open Research Software* **7**(1).

Hothorn, T. and Zeileis, A. (2015). partykit: a modular toolkit for recursive partytioning in R. *The Journal of Machine Learning Research* **16**(1), 3905-3909.

See Also

`dipm`, `spmtree`

Examples

```
#' #
# ... an example with a continuous outcome variable
#     and two treatment groups
#

N = 100
set.seed(123)

# generate binary treatments
treatment = rbinom(N, 1, 0.5)
```

```
# generate candidate split variables
X1 = rnorm(n = N, mean = 0, sd = 1)
X2 = rnorm(n = N, mean = 0, sd = 1)
X3 = rnorm(n = N, mean = 0, sd = 1)
X4 = rnorm(n = N, mean = 0, sd = 1)
X5 = rnorm(n = N, mean = 0, sd = 1)
X = cbind(X1, X2, X3, X4, X5)
colnames(X) = paste0("X", 1:5)

# generate continuous outcome variable
calculateLink = function(X, treatment){

    ((X[, 1] <= 0) & (X[, 2] <= 0)) *
        (25 * (1 - treatment) + 8 * treatment) +

    ((X[, 1] <= 0) & (X[, 2] > 0)) *
        (18 * (1 - treatment) + 20 * treatment) +

    ((X[,1 ] > 0) & (X[, 3] <= 0)) *
        (20 * (1 - treatment) + 18 * treatment) +

    ((X[,1] > 0) & (X[,3] > 0)) *
        (8 * (1 - treatment) + 25 * treatment)
}

Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# fit a dipm classification tree
tree = dipm(Y ~ treatment | ., data, mtry = 1, maxdepth = 3)
plot(tree, terminal_panel = node_dipm)
```

pmprune *Pruning A Precision Medicine Tree*

Description

This function prunes classification trees designed for the precision medicine setting.

Usage

```
pmprune(tree)
```

Arguments

`tree`	A data frame object returned from either the `dipm()` or `spmtree()` function

Details

This function implements the simple pruning strategy proposed and used in Tsai et al. (2016). Terminal sister nodes, i.e., nodes with no child nodes that share the same parent node, are removed if they have the same identified optimal treatment assignment.

Value

`pmprune` returns the pruned classification tree as a data frame. The data frame contains the following columns of information:

`node`	Unique integer values that identify each node in the tree, where all of the nodes are indexed starting from 1
`splitvar`	Integers that represent the candidate split variable used to split each node, where all of the variables are indexed starting from 1; for terminal nodes, i.e., nodes without child nodes, the value is set equal to NA
`splitvar_name`	The names of the candidate split variables used to split each node obtained from the column names of the supplied data; for terminal nodes, the value is set equal to NA
`type`	Characters that denote the type of each candidate split variable; "bin" is for binary variables, "ord" for ordinal, and "nom" for nominal; for terminal nodes, the value is set equal to NA
`splitval`	Values of the left child node of the current split/node; for binary variables, a value of 0 is printed, and subjects with values of 0 for the current `splitvar` are in the left child node, while subjects with values of 1 are in the right child node; for ordinal variables, `splitval` is numeric and implies that subjects with values of the current `splitvar` less than or equal to `splitval` are in the left child node, while the remaining subjects with values greater than `splitval` are in the right child node; for nominal variables, the `splitval` is a set of integers separated by commas, and subjects in that set of categories are in the left child node, while the remaining subjects are in the right child node; for terminal nodes, the value is set equal to NA
`lchild`	Integers that represent the index (i.e., `node` value) of each node's left child node; for terminal nodes, the value is set equal to NA
`rchild`	Integers that represent the index (i.e., `node` value) of each node's right child node; for terminal nodes, the value is set equal to NA
`depth`	Integers that specify the depth of each node; the root node has depth 1, its children have depth 2, etc.
`nsubj`	Integers that count the total number of subjects within each node
`besttrt`	Integers that denote the identified best treatment assignment of each node

References

Tsai, W.-M., Zhang, H., Buta, E., O'Malley, S., Gueorguieva, R. (2016). A modified classification tree method for personalized medicine decisions. *Statistics and its Interface* **9**, 239-253.

See Also

dipm, spmtree

Examples

```
#
# ... an example with a continuous outcome variable
#     and three treatment groups
#


N = 100
set.seed(123)

# generate treatments
treatment = sample(1:3, N, replace = TRUE)

# generate candidate split variables
X1 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X2 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X3 = sample(1:4, N, replace = TRUE)
X4 = sample(1:5, N, replace = TRUE)
X5 = rbinom(N, 1, 0.5)
X6 = rbinom(N, 1, 0.5)
X7 = rbinom(N, 1, 0.5)
X = cbind(X1, X2, X3, X4, X5, X6, X7)
colnames(X) = paste0("X", 1:7)

# generate continuous outcome variable
calculateLink = function(X, treatment){

    10.2 - 0.3 * (treatment == 1) - 0.1 * X[, 1] +
    2.1 * (treatment == 1) * X[, 1] +
    1.2 * X[, 2]
}

Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# create vector of variable types
types = c(rep("ordinal", 2), rep("nominal", 2), rep("binary", 3),
            "response", "treatment")

# fit a classification tree
tree = spmtree(Y ~ treatment | ., data, types = types, dataframe = TRUE)

# prune the tree
ptree = pmprune(tree)
```

spmtree — *Simple Precision Medicine Tree*

Description

This function creates a classification tree designed to identify subgroups in which subjects perform especially well or especially poorly in a given treatment group.

Usage

```
spmtree(
  formula,
  data,
  types = NULL,
  nmin = 5,
  maxdepth = Inf,
  print = TRUE,
  dataframe = FALSE,
  prune = FALSE
)
```

Arguments

`formula`	A description of the model to be fit with format `Y ~ treatment | X1 + X2` for data with a continuous outcome variable Y and `Surv(Y, delta) ~ treatment | X1 + X2` for data with a right-censored survival outcome variable Y and a status indicator delta
`data`	A matrix or data frame of the data
`types`	A vector, data frame, or matrix of the types of each variable in the data; if left blank, the default is to assume all of the candidate split variables are ordinal; otherwise, all variables in the data must be specified, and the possible variable types are: "response", "treatment", "status", "binary", "ordinal", and "nominal" for outcome variable Y, the treatment variable, the status indicator (if applicable), binary candidate split variables, ordinal candidate split variables, and nominal candidate split variables respectively
`nmin`	An integer specifying the minimum node size of the overall classification tree
`maxdepth`	An integer specifying the maximum depth of the overall classification tree; this argument is optional but useful for shortening computation time; if left blank, the default is to grow the full tree until the minimum node size `nmin` is reached
`print`	A boolean (TRUE/FALSE) value, where TRUE prints a more readable version of the final tree to the screen
`dataframe`	A boolean (TRUE/FALSE) value, where TRUE returns the final tree as a dataframe
`prune`	A boolean (TRUE/FALSE) value, where TRUE prunes the final tree using `pmprune` function

Details

To identify the best split at each node of the classification tree, all possible splits of all candidate split variables are considered. The single split with the highest split criteria score is identified as the best split of the node. For data with a continuous outcome variable, the split criteria is the DIFF value that was first proposed for usage in the relative-effectiveness based method (Zhang et al. (2010), Tsai et al. (2016)). For data with a survival outcome variable, the split criteria is the squared test statistic that tests the significance of the split by treatment interaction term in a Cox proportional hazards model.

When using `spmtree`, note the following requirements for the supplied data. First, the dataset must contain an outcome variable Y and a treatment variable. If Y is a right-censored survival time outcome, then there must also be a status indicator delta, where values of 1 denote the occurrence of the (harmful) event of interest, and values of 0 denote censoring. If there are only two treatment groups, then the two possible values must be 0 or 1. If there are more than two treatment groups, then the possible values must be integers starting from 1 to the total number of treatment assignments. In regard to the candidate split variables, if a variable is binary, then the variable must take values of 0 or 1. If a variable is nominal, then the values must be integers starting from 1 to the total number of categories. There cannot be any missing values in the dataset. For candidate split variables with missing values, the missings together (MT) method proposed by Zhang et al. (1996) is helpful.

Value

`spmtree` returns the final classification tree as a `party` object by default or a data frame. See Hothorn and Zeileis (2015) for details. The data frame contains the following columns of information:

`node`	Unique integer values that identify each node in the tree, where all of the nodes are indexed starting from 1
`splitvar`	Integers that represent the candidate split variable used to split each node, where all of the variables are indexed starting from 1; for terminal nodes, i.e., nodes without child nodes, the value is set equal to NA
`splitvar_name`	The names of the candidate split variables used to split each node obtained from the column names of the supplied data; for terminal nodes, the value is set equal to NA
`type`	Characters that denote the type of each candidate split variable; "bin" is for binary variables, "ord" for ordinal, and "nom" for nominal; for terminal nodes, the value is set equal to NA
`splitval`	Values of the left child node of the current split/node; for binary variables, a value of 0 is printed, and subjects with values of 0 for the current `splitvar` are in the left child node, while subjects with values of 1 are in the right child node; for ordinal variables, `splitval` is numeric and implies that subjects with values of the current `splitvar` less than or equal to `splitval` are in the left child node, while the remaining subjects with values greater than `splitval` are in the right child node; for nominal variables, the `splitval` is a set of integers separated by commas, and subjects in that set of categories are in the left child node, while the remaining subjects are in the right child node; for terminal nodes, the value is set equal to NA

`lchild`	Integers that represent the index (i.e., `node` value) of each node's left child node; for terminal nodes, the value is set equal to NA
`rchild`	Integers that represent the index (i.e., `node` value) of each node's right child node; for terminal nodes, the value is set equal to NA
`depth`	Integers that specify the depth of each node; the root node has depth 1, its children have depth 2, etc.
`nsubj`	Integers that count the total number of subjects within each node
`besttrt`	Integers that denote the identified best treatment assignment of each node

References

Chen, V., Li, C., and Zhang, H. (2022). dipm: an R package implementing the Depth Importance in Precision Medicine (DIPM) tree and Forest-based method. *Bioinformatics Advances*, **2**(1), vbac041.

Chen, V. and Zhang, H. (2022). Depth importance in precision medicine (DIPM): A tree-and forest-based method for right-censored survival outcomes. *Biostatistics* **23**(1), 157-172.

Chen, V. and Zhang, H. (2020). Depth importance in precision medicine (DIPM): a tree and forest based method. In *Contemporary Experimental Design, Multivariate Analysis and Data Mining*, 243-259.

Tsai, W.-M., Zhang, H., Buta, E., O'Malley, S., Gueorguieva, R. (2016). A modified classification tree method for personalized medicine decisions. *Statistics and its Interface* **9**, 239-253.

Zhang, H., Holford, T., and Bracken, M.B. (1996). A tree-based method of analysis for prospective studies. *Statistics in Medicine* **15**, 37-49.

Zhang, H., Legro, R.S., Zhang, J., Zhang, L., Chen, X., et al. (2010). Decision trees for identifying predictors of treatment effectiveness in clinical trials and its application to ovulation in a study of women with polycystic ovary syndrome. *Human Reproduction* **25**, 2612-2621.

Hothorn, T. and Zeileis, A. (2015). partykit: a modular toolkit for recursive partytioning in R. *The Journal of Machine Learning Research* **16**(1), 3905-3909.

See Also

`dipm`

Examples

```
#
# ... an example with a continuous outcome variable
#     and two treatment groups
#

N = 300
set.seed(123)

# generate binary treatments
treatment = rbinom(N, 1, 0.5)
```

```
# generate candidate split variables
X1 = rnorm(n = N, mean = 0, sd = 1)
X2 = rnorm(n = N, mean = 0, sd = 1)
X3 = rnorm(n = N, mean = 0, sd = 1)
X4 = rnorm(n = N, mean = 0, sd = 1)
X5 = rnorm(n = N, mean = 0, sd = 1)
X = cbind(X1, X2, X3, X4, X5)
colnames(X) = paste0("X", 1:5)

# generate continuous outcome variable
calculateLink = function(X, treatment){

    ((X[, 1] <= 0) & (X[, 2] <= 0)) *
        (25 * (1 - treatment) + 8 * treatment) +

    ((X[, 1] <= 0) & (X[, 2] > 0)) *
        (18 * (1 - treatment) + 20 * treatment) +

    ((X[, 1] > 0) & (X[, 3] <= 0)) *
        (20 * (1 - treatment) + 18 * treatment) +

    ((X[, 1] > 0) & (X[, 3] > 0)) *
        (8 * (1 - treatment) + 25 * treatment)
}

Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# fit a classification tree
tree1 = spmtree(Y ~ treatment | ., data, maxdepth = 3)
# predict optimal treatment for new subjects
predict(tree1, newdata = head(data),
FUN = function(n)  as.numeric(n$info$opt_trt))


#
# ... an example with a continuous outcome variable
#     and three treatment groups
#

N = 600
set.seed(123)

# generate treatments
treatment = sample(1:3, N, replace = TRUE)

# generate candidate split variables
X1 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X2 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X3 = sample(1:4, N, replace = TRUE)
```

```
X4 = sample(1:5, N, replace = TRUE)
X5 = rbinom(N, 1, 0.5)
X6 = rbinom(N, 1, 0.5)
X7 = rbinom(N, 1, 0.5)
X = cbind(X1, X2, X3, X4, X5, X6, X7)
colnames(X) = paste0("X", 1:7)

# generate continuous outcome variable
calculateLink = function(X, treatment){

    10.2 - 0.3 * (treatment == 1) - 0.1 * X[, 1] +
    2.1 * (treatment == 1) * X[, 1] +
    1.2 * X[, 2]
}

Link = calculateLink(X, treatment)
Y = rnorm(N, mean = Link, sd = 1)

# combine variables in a data frame
data = data.frame(X, Y, treatment)

# create vector of variable types
types = c(rep("ordinal", 2), rep("nominal", 2), rep("binary", 3),
        "response", "treatment")

# fit a classification tree
tree2 = spmtree(Y ~ treatment | ., data, types = types)

#
# ... an example with a survival outcome variable
#     and two treatment groups
#

N = 300
set.seed(321)

# generate binary treatments
treatment = rbinom(N, 1, 0.5)

# generate candidate split variables
X1 = rnorm(n = N, mean = 0, sd = 1)
X2 = rnorm(n = N, mean = 0, sd = 1)
X3 = rnorm(n = N, mean = 0, sd = 1)
X4 = rnorm(n = N, mean = 0, sd = 1)
X5 = rnorm(n = N, mean = 0, sd = 1)
X = cbind(X1, X2, X3, X4, X5)
colnames(X) = paste0("X", 1:5)

# generate survival outcome variable
calculateLink = function(X, treatment){

    X[, 1] + 0.5 * X[, 3] + (3 * treatment - 1.5) * (abs(X[, 5]) - 0.67)
}
```

```
Link = calculateLink(X, treatment)
T = rexp(N, exp(-Link))
C0 = rexp(N, 0.1 * exp(X[, 5] + X[, 2]))
Y = pmin(T, C0)
delta = (T <= C0)

# combine variables in a data frame
data = data.frame(X, Y, delta, treatment)

# fit a classification tree
tree3 = spmtree(Surv(Y, delta) ~ treatment | ., data, maxdepth = 2)

#
# ... an example with a survival outcome variable
#     and four treatment groups
#

N = 800
set.seed(321)

# generate treatments
treatment = sample(1:4, N, replace = TRUE)

# generate candidate split variables
X1 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X2 = round(rnorm(n = N, mean = 0, sd = 1), 4)
X3 = sample(1:4, N, replace = TRUE)
X4 = sample(1:5, N, replace = TRUE)
X5 = rbinom(N, 1, 0.5)
X6 = rbinom(N, 1, 0.5)
X7 = rbinom(N, 1, 0.5)
X = cbind(X1, X2, X3, X4, X5, X6, X7)
colnames(X) = paste0("X", 1:7)

# generate survival outcome variable
calculateLink = function(X, treatment, noise){

    -0.2 * (treatment == 1) +
    -1.1 * X[, 1] +
    1.2 * (treatment == 1) * X[, 1] +
    1.2 * X[, 2]
}

Link = calculateLink(X, treatment)
T = rweibull(N, shape = 2, scale = exp(Link))
Cnoise = runif(n = N) + runif(n = N)
C0 = rexp(N, exp(0.3 * -Cnoise))
Y = pmin(T, C0)
delta = (T <= C0)

# combine variables in a data frame
data = data.frame(X, Y, delta, treatment)
```

```
# create vector of variable types
types = c(rep("ordinal", 2), rep("nominal", 2), rep("binary", 3),
        "response", "status", "treatment")

# fit two classification trees
tree4 = spmtree(Surv(Y, delta) ~ treatment | ., data, types = types, maxdepth = 2)
tree5 = spmtree(Surv(Y, delta) ~ treatment | X3 + X4, data, types = types,
             maxdepth = 2)
```

Index