Package: TSDT (via r-universe)

September 14, 2024

Type Package

Title Treatment-Specific Subgroup Detection Tool

Version 1.0.7

Date 2022-04-06

Description Implements a method for identifying subgroups with superior response relative to the overall sample.

Imports methods, mlbench, hash, party, rpart, survival, survRM2, stats, modeltools, utils, parallel

LazyLoad no

License GPL (>= 2)

URL https://github.com/EliLillyCo/CRAN_TSDT

BugReports https://github.com/EliLillyCo/CRAN_TSDT/issues

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Chakib Battioui [aut], Brian Denton [aut, cre], Lei Shen [ctb], Eli Lilly and Company [cph]

Maintainer Brian Denton <denton_brian_david@lilly.com>

Date/Publication 2022-04-06 22:22:28 UTC

Repository https://dentonbriandavid.r-universe.dev

RemoteUrl https://github.com/cran/TSDT

RemoteRef HEAD

RemoteSha af0cdf44a15b77661be4490d4f11ac1f04c3ece5

Contents

binary_transform *binary transform*

Description

Converts any variable with two possible values to a 0,1 binary variable.

Usage

```
binary_transform(x)
```

Arguments

x	A variable with two possible values.

Value

A vector with values in 0,1.

Examples

```
## Convert a variable that takes values 'A' and 'B' to 0 and 1
x <- sample( c('A','B'), size = 10, prob = c(0.5,0.5), replace = TRUE )
print(x);flush.console()
binary_transform( x )
```

bootstrap *bootstrap*

Description

Generate a vector of bootstrap samples.

Usage

```
bootstrap(
  x,
  trt = NULL,
  trt_control = "Control",
  FUN = NULL,
  varname = NULL,
  varcol = NULL,
  arglist = NULL,
  n_samples = 1
)
```

Arguments

x	Source data to bootstrap.
trt	Treatment variable. (optional)
trt_control	Value for treatment control arm. Default value is 'Control'.
FUN	Function to compute statistic for each bootstrap sample. (optional)
varname	Name of variable in x on which to compute FUN. If x has only one column varname is not needed. If x has more than one column then either varname or varcol must be specified.
varcol	Column index of x on which to compute FUN. If x has only one column varcol is not needed. If x has more than one column then either varname or varcol must be specified.
arglist	List of additional arguments to pass to FUN.
n_samples	Number of bootstrap samples to generate.

Details

Each bootstrap sample will retain the in-bag and out-of-bag data. Optionally, the user may specify a function to compute a statistic for each in-bag and out-of-bag sample. This function may be a built-in R function (e.g. mean, median, etc.) or a user-defined function (see Examples). If no statistic function is provided bootstrap returns a vector of objects of class Bootstrap. If a statistic function is provided bootstrap returns a vector of objects of class BootstrapStatistic, which in addition to the in-bag and out-of-bag samples contains the name of the statistic, variable on which the statistic is computed, and the numerical result of the statistic for each in-bag and out-of-bag sample.

Value

If FUN is NULL returns a vector of objects of class Bootstrap. If FUN is non-NULL returns a vector of objects of class BootstrapStatistic

See Also

Bootstrap, BootstrapStatistic

Examples

```
## Generate example data frame containing response and treatment
N <- 20
x <- data.frame( runif( N ) )
names( x ) <- "response"
x$treatment <- factor( sample( c("Control","Experimental"), size = N,
                       prob = c(0.8,0.2), replace = TRUE ) )

## Generate two bootstrap samples without regard to treatment
ex1 <- bootstrap( x, n_samples = 2 )

## Generate two bootstrap samples stratified by treatment
ex2 <- bootstrap( x, trt = x$treatment, trt_control = "Control", n_samples = 2 )
```

```
## For each bootstrap sample compute a statistic on the in-bag and out-of-bag data
ex3 <- bootstrap( x, FUN = mean, varname = "response", n_samples = 2 )

## Specify a user-defined function that takes a numeric vector input and
## returns a numeric result
sort_and_rank <- function( z, rank ){
  z <- sort( z )
  return( z[rank] )
}

ex4 <- bootstrap( x, FUN = sort_and_rank, arglist = list( rank = 1 ),
                  varname = "response", n_samples = 2 )
```

`Bootstrap-class` *Bootstrap*

Description

Bootstrap is a container class for bootstrap samples.

Value

Object of class Bootstrap

Slots

`inbag` In-bag bootstrap sample.

`oob` Out-of-bag bootstrap sample.

See Also

BootstrapStatistic, bootstrap

`BootstrapStatistic-class` *BootstrapStatistic*

Description

BootstrapStatistic is a container class for bootstrap samples augmented with a computed statistic.

Value

Object of class BootstrapStatistic

Slots

`statname` The name of a (possibly user-defined) statistic to compute on the bootstrap sample.

`arglist` A list of arguments passed to the function referenced by statname.

`variable` The name of the variable on which to compute statname.

`inbag_stat` The value of statname for the in-bag bootstrapped sample.

`oob_stat` The value of statname for the out-of-bag bootstrapped sample.

See Also

Bootstrap, bootstrap

`CTree-class` *CTree*

Description

CTree is a container class for trees created by ctree.

Value

An object of class CTree

Slots

`tree` An object of class BinaryTree-class produced by ctree.

`data` Training data.

`parameters` Control parameters

See Also

ctree, BinaryTree-class

ctree_wrapper *ctree_wrapper*

Description

A wrapper function to ctree

Usage

```
ctree_wrapper(response, covariates = NULL, tree_builder_parameters = list())
```

Arguments

response	Response variable to use in ctree model.
covariates	Covariates to use in ctree model.
tree_builder_parameters	A named list of parameters to pass to ctree.

Value

An object of class CTree

See Also

ctree

Examples

```
requireNamespace( "party", quietly = TRUE )
## From party::ctree() examples:
set.seed(290875)
airq <- subset(airquality, !is.na(Ozone))

## Provide response and covariates to fit ctree
ex1 <- ctree_wrapper( response = airq$Ozone,
                      covariates = subset( airq, select = -Ozone ) )

## Pass list of control parameters. Note that ctree takes a parameter called
## 'controls' (with an 's'), rather than 'control' as in rpart.
ex2 <- ctree_wrapper( response = airq$Ozone,
                      covariates = subset( airq, select = -Ozone ),
                      tree_builder_parameters = list( controls =
                                          party::ctree_control( maxdepth = 2 ) ) )
```

`cutpoints` *Get distribution of cutpoints for subgroups.*

Description

Get distribution of cutpoints for subgroups.

Usage

```
cutpoints(object, subgroup = NULL, subsub = NULL)
```

Arguments

`object`	An object of class TSDT
`subgroup`	A string decscription of a subgroup (optional)
`subsub`	A string description of a sub-subgroup (optional)

Value

A vector containing the subgroup cutpoints.

See Also

TSDT

`diff_mean_deviance_residuals` *diff_mean_deviance_residuals*

Description

Computes the difference in the mean of deviance residuals function across treatment groups.

Usage

```
diff_mean_deviance_residuals(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

The deviance residual is the observed number of events at time t minus the expected number of events at time t. See documentation for mean_deviance_residuals (linked below) for more details. A smaller value for the deviance residual is preferred when the event under study is an undesirable event – i.e. it is preferred to observe fewer events than predicted by the survival model. A two-arm TSDT model computes the mean deviance residual in the treatment arm minus the mean deviance residual in the control arm. The treatment arm is superior to the control arm when the mean deviance residual in the treatment arm is less than the mean deviance residual in the control arm. Thus, the appropriate value for desirable_response is desirable_response = 'decreasing'. If the event under study is a desirable event the appropriate value for desirable_response is desirable_response = 'increasing'. It is assumed most survival models will model an undesirable event, so the default value for desirable_response when the scoring_function is diff_mean_deviance_residuals is desirable_response = 'decreasing'. Note this differs from all other TSDT configurations, for which the default value for desirable_response is desirable_response = 'increasing'.

Value

Difference in mean deviance residuals across treatment arms.

See Also

mean_deviance_residuals, Surv, coxph, survreg, residuals.coxph, residuals.survreg, TSDT

`diff_quantile_response` *diff_quantile_response*

Description

Return the difference across treatment arms of a specified response quantile

Usage

```
diff_quantile_response(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

This function returns the difference across treatment arms of the response quantile associated with a specified percentile. The default behavior is to return the difference in medians.

Value

A difference of response quantiles across treatment arms

See Also

TSDT, quantile_response, quantile

Examples

```
## Generate example data containing response and treatment
N <- 100
y = runif( min = 0, max = 20, n = N )
df <- as.data.frame( y )
names( df )  <- "y"
df$trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6),
                  replace = TRUE )

## Default behavior is to return the median
diff_quantile_response( df )

# should match previous result from quantile_response
median( df$y[df$trt!='Control'] ) - median( df$y[df$trt=='Control'] )

## Get Q1 response
diff_quantile_response( df, scoring_function_parameters = list( percentile = 0.25 ) )

# should match previous result from quantile_response
quantile( df$y[df$trt!='Control'], 0.25 ) - quantile( df$y[df$trt=='Control'], 0.25 )

## Get max response
diff_quantile_response( df, scoring_function_parameters = list( percentile = 1 ) )

# should match previous result from quantile_response
max( df$y[df$trt!='Control'] ) -  max( df$y[df$trt=='Control'] )
```

---

diff_restricted_mean_survival_time

diff_restricted_mean_survival_time

Description

Computes the difference in restricted mean survival time across treatment arms.

Usage

```
diff_restricted_mean_survival_time(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

Computes the restricted mean survival time for the treatment and control arms and returns the difference.

Value

Difference in restricted mean survival time across treatment arms.

See Also

TSDT, Surv, rmst2

`diff_survival_time_quantile` *diff_survival_time_quantile*

Description

Computes the difference in the quantile of a survival function across treatment groups.

Usage

```
diff_survival_time_quantile(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

Computes the survival function quantile for the treatment and control arms and returns the difference.

Value

A difference in a survival time quantile across treatment arms.

See Also

TSDT, survival_time_quantile, Surv, coxph, survfit, survreg, predict.coxph, predict.survreg

Examples

```
requireNamespace( "survival", quiet = TRUE )
N <- 200
df <- data.frame( y = survival::Surv( runif( min = 0, max = 20, n = N ),
                        sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE ) ),
                  trt = sample( c('Control','Experimental'), size = N,
                                prob = c(0.4,0.6), replace = TRUE ) )

## Compute difference in median survival time between Experimental arm and
## Control arm.  It is not actually necessary to provide the value for the
## time_var, trt_var, trt_control, and percentile parameters because these
## values are all equal to their default values. The value are explicitly
## provided here simply for clarity.
ex1 <- diff_survival_time_quantile( data = df,
                                    scoring_function_parameters = list( trt_var = "trt",
                                    trt_control = "Control",
                                    percentile = 0.50 ) )

## Compute difference in Q1 survival time. In this example the default value
## for all scoring function parameters are used except percentile, which here
## takes the value 0.25.
ex2 <- diff_survival_time_quantile( data = df,
                                scoring_function_parameters = list( percentile = 0.25 ) )
```

`distribution` *distribution*

Description

Returns the distribution of values used to compute TSDT summary statistics.

Usage

```
distribution(object, statistic, subgroup = NULL, subsub = NULL)
```

Arguments

`object`	An object of class TSDT
`statistic`	The desired statistic distribution
`subgroup`	The desired subgroup
`subsub`	A subset of the subgroup

Details

This function returns the distribution of all values used to compute summary statistics for superior subgroups identified by the TSDT algorithm. The summary statistics returned for a TSDT object include the mean subgroup size, mean response value, and median value of the scoring function. These statistics reported seperately for in-bag and out-of-bag data sets, and also stratified by treatment arm. This function can also provide the distribution of all cutpoints for a numeric splitting variable in a subgroup definition.

Value

A vector containing the observed values for the specified subgroup

See Also

TSDT, summary-methods

Examples

```
set.seed(0)
N <- 200
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6),
               replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )
covariates <- data.frame( X1 )
covariates$X2 <- X2
covariates$X3 <- factor( X3 )
covariates$X4 <- factor( X4 )

## Create a TSDT object
ex1 <- TSDT( response = continuous_response,
            trt = trt, trt_control = 'Control',
            covariates = covariates[,1:4],
            inbag_score_margin = 0,
            desirable_response = "increasing",
            oob_score_margin = 0,
            min_subgroup_n_control = 5,
            min_subgroup_n_trt = 5,
            n_sample = 5 )

## Show summary statistics
summary( ex1 )

## Get the number of subjects in each superior in-bag subgroup
distribution( ex1, statistic = 'Inbag_Subgroup_Size' )

## Get the vector of subgroup sample sizes for a particular subgroup
distribution( ex1, statistic = 'Inbag_Subgroup_Size',
              subgroup = 'X1<xxxxx & X1>=xxxxx' )

## Get the observed cutpoints for the numeric splitting variables in a subgroup
distribution( ex1, statistic = 'Cutpoints', subgroup = 'X1<xxxxx & X1>=xxxxx' )

## If the subgroup definition has more than one numeric splitting variable you
## can retrieve the numeric cutpoints for the splitting variables individually
distribution( ex1, statistic = 'Cutpoints', subgroup = 'X1<xxxxx & X1>=xxxxx',
              subsub = 'X1<xxxxx' )
distribution( ex1, statistic = 'Cutpoints', subgroup = 'X1<xxxxx & X1>=xxxxx',
```

```
              subsub = 'X1>=xxxxx' )

## Valid statistic names come from the column names in the summary output. If
## you are uncertain what the possible statistic values could be, you can pass
## any arbitrary string as the statistic and an error message is returned
## listing valid statistic values.
## Not run:
distribution( ex1, statistic = 'Invalid_Statistic' )

## End(Not run)
```

`folds` *folds*

Description

Partition data into k folds for k-fold cross-validation. Adds a variable fold_id to the data.frame.

Usage

```
folds(x, k)
```

Arguments

x	data.frame to partition into k folds for k-fold cross-validation.
k	Number of folds to use in cross-validation

Value

A list of partitions of the vector x.

Examples

```
# Generate random example data
N <- 200
ID <- 1:N
continuous_response = runif( min = 0, max = 20, n = N )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )

df <- data.frame( ID )
names( df ) <- "ID"
df$response <- continuous_response
df$X1 <- X1
df$X2 <- X2
df$X3 <- factor( X3 )
df$X4 <- factor( X4 )
```

```
## Partition data into 5 folds
ex1 <- folds( df, k = 5 )

## Partition data into 10 folds
ex2 <- folds( df, k = 10 )
```

`function_parameter_names` *function_parameter_names*

Description

Returns a character vector of the specified function's parameters

Usage

```
function_parameter_names(FUN)
```

Arguments

FUN	The name of a function

Value

A character vector of function parameter names

Examples

```
## Define a function
example_function <- function( parm1, arg2, x, bool = FALSE ){
  cat( "This is an example function.\n" )
}

## Return the function parameter names
function_parameter_names( example_function )
```

`get_covariates` *get_covariates*

Description

Returns the covariate variables in the in-bag or out-of-bag data.

Usage

```
get_covariates(data, scoring_function_parameters)
```

Arguments

data	A data.frame containing in-bag or out-of-bag data
scoring_function_parameters	A list of named elements containing control parameters and other data required by the scoring function

Details

If the user provides a covariate_vars parameter in the list of scoring_function_parameters this function will return the variables specified by that parameter. If the user specifies a covariate_cols parameter in the list of scoring_function_parameters the function returns the columns in data indexed by that parameter. Otherwise, NULL is returned.

Value

A data.frame of covariates.

See Also

get_y, get_trt

Examples

```
## Create an example data.frame
df <- data.frame( y <- 1:5 )
names( df ) <- "y"
df$time <- 10:14
df$time2 <- 20:24
df$event <- sample( c(0:1), size = 5, replace = TRUE )
df$trt <- sample( c("Control","Treatment"), size = 5, replace = TRUE )
df$x1 <- runif( n = 5 )
df$x2 <- LETTERS[1:5]

## Select the covariate variables by name
get_covariates( df, scoring_function_parameters = list( covariate_vars = c("x1","x2") ) )

## Select the covariate variables by column index
get_covariates( df, scoring_function_parameters = list( covariate_cols = c(6:7) ) )
```

---

get_cutpoints *get_cutpoints*

Description

Accessor method for cutpoints slot in TSDT objects.

Usage

```
get_cutpoints(.Object, subgroup, subsub = NULL)

## S4 method for signature 'TSDT_CutpointDistribution'
get_cutpoints(.Object, subgroup = character, subsub = NULL)

## S4 method for signature 'TSDT'
get_cutpoints(.Object, subgroup = character, subsub = NULL)
```

Arguments

.Object	A TSDT object.
subgroup	The anonymized subgroup.
subsub	A particular component of the subgroup to retrieve.

Details

The summary results from TSDT provide a set of 'anonymized' subgroups in a form similar to 'X1<xxxxx'. The variable X1 may have been selected as a splitting variable in several bootstrapped samples. The exact numerical cutpoint for X1 could vary from one sample to the next. The get_cutpoints method returns all the numerical cutpoints associated with this subgroup. If the subgroup is a compound subgroup defined on more than one spliting variable the user can specify the 'subsub' parameter to get the cutpoints associated with a particular component of the subgroup.

Examples

```
## Not run:
example( TSDT )
## You can access the cutpoints slot of a TSDT object directly
ex2@cutpoints

## You can also use the accessor method
get_cutpoints( ex2@cutpoints, subgroup = 'X1<xxxxx' )

## Retrieving a compound subgroup defined on multiple splits
get_cutpoints( ex2, subgroup = 'X1<xxxxx & X1>=xxxxx' )

## Retrieving a single component from the compound subgroup
get_cutpoints( ex2, subgroup = 'X1<xxxxx & X1>=xxxxx', subsub = 'X1>=xxxxx' )

## End(Not run)
```

---

get_suggested_subgroup *get_suggested_subgroup*

Description

Get a string definition of the suggested subgroup definition.

Usage

```
get_suggested_subgroup(anonymized_subgroup, suggested_cutoff, anon = "xxxxx")
```

Arguments

anonymized_subgroup	A string containing the the anonymized subgroup.
suggested_cutoff	A string containing the suggested cutoff.
anon	The anonymization string. By default this is 'xxxxx'.

Details

Subgroups are reported in an anonymized fashion – e.g. a subgroup defined on a variable X1 could be reported as X1<xxxxx, 'xxxxx' is a string used to represent an exact numeric cutoff. For each anonymized subgroup, the distribution of exact numeric cutpoints is retained across all bootrstrapped samples. TSDT then provides a suggested cutoff got each anonymized subgroup. By default, this suggested cutoff is the median of the observed cutpoints. Note that this anonymization applies only to numeric splitting variables. Categorical splitting variables are not anonymized.

Examples

```
set.seed(0)
N <- 200
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )
covariates <- data.frame( X1 )
covariates$X2 <- X2
covariates$X3 <- factor( X3 )
covariates$X4 <- factor( X4 )

## Create a TSDT object
ex1 <- TSDT( response = continuous_response,
             trt = trt, trt_control = 'Control',
             covariates = covariates[,1:4],
             inbag_score_margin = 0,
             desirable_response = "increasing",
             oob_score_margin = 0,
             min_subgroup_n_control = 10,
             min_subgroup_n_trt = 20,
             maxdepth = 2,
             rootcompete = 2 )
```

```
## Show summary statistics
summary( ex1 )

## Get the anonymized subgroup defined on X1
anonymized_subgroup <- as.character( ex1@superior_subgroups$Subgroup[2] )

## Get the suggested cutoff for this subgroup
suggested_cutoff <- as.character( ex1@superior_subgroups$Suggested_Cutoff[2] )

## Get the suggested subgroup
get_suggested_subgroup( anonymized_subgroup = anonymized_subgroup,
                        suggested_cutoff = suggested_cutoff )
```

`get_trt` *get_trt*

Description

Returns the treatment variable in the in-bag or out-of-bag data.

Usage

```
get_trt(data, scoring_function_parameters = NULL)
```

Arguments

`data`	A data.frame containing in-bag or out-of-bag data
`scoring_function_parameters`	A list of named elements containing control parameters and other data required by the scoring function

Details

If the user provides a trt_var parameter in the list of scoring_function_parameters this function will return the variable specified by that parameter. If the user specifies a trt_col parameter in the list of scoring_function_parameters the function returns the column in data indexed by that parameter. Lastly, if data contains a variable called 'trt' that variable is returned. Otherwise, NULL is returned.

Value

Treatment variable (if available) or NULL.

See Also

get_y, get_covariates

Examples

```
## Create an example data.frame
df <- data.frame( y <- 1:5 )
names( df ) <- "y"
df$time <- 10:14
df$time2 <- 20:24
df$event <- sample( c(0:1), size = 5, replace = TRUE )
df$trt <- sample( c("Control","Treatment"), size = 5, replace = TRUE )
df$x1 <- runif( n = 5 )
df$x2 <- LETTERS[1:5]

## Select the trt variable by name
get_trt( df, scoring_function_parameters = list( trt_var = 'trt' ) )

## Select the trt variable by column index
get_trt( df, scoring_function_parameters = list( trt_col = 5 ) )

## The default behavior works for this example because the trt variable in df
## is actually called trt.
get_trt( df )

## If the user's data does not contain a variable called
## 'y' the default behavior will fail. In this case the user must explicitly
## identify the 'y' variable via one of the two previous methods.
names( df )[which(names(df) == "trt")] <- "treatment" # rename the 'trt' variable to 'treatment'

get_trt( df )  # now default behavior fails (i.e. returns NULL)

get_trt( df, scoring_function_parameters = list( trt_var = 'treatment' ) ) # this works
```

---

get_y *get_y*

Description

Returns the response variable in the in-bag or out-of-bag data.

Usage

```
get_y(data, scoring_function_parameters = NULL)
```

Arguments

`data`	A data.frame containing in-bag or out-of-bag data
`scoring_function_parameters`	A list of named elements containing control parameters and other data required by the scoring function

Details

If the user provides a y_var parameter in the list of scoring_function_parameters this function will return the variable specified by that parameter. If the user specifies a y_col parameter in the list of scoring_function_parameters the function returns the column in data indexed by that parameter. Lastly, if data contains a variable called 'y' that variable is returned. Otherwise, NULL is returned.

Value

Response variable (if present) or NULL.

See Also

get_trt, get_covariates

Examples

```
## Create an example data.frame
df <- data.frame( y <- 1:5 )
names( df ) <- "y"
df$time <- 10:14
df$time2 <- 20:24
df$event <- sample( c(0:1), size = 5, replace = TRUE )
df$trt <- sample( c("Control","Treatment"), size = 5, replace = TRUE )
df$x1 <- runif( n = 5 )
df$x2 <- LETTERS[1:5]

## Select the y variable by name
get_y( df, scoring_function_parameters = list( y_var = 'y' ) )

## Select the y variable by column index
get_y( df, scoring_function_parameters = list( y_col = 1 ) )

## The default behavior works for this example because the y variable in df
## is actually called y.
get_y( df )

## If the user's data does not contain a variable called
## 'y' the default behavior will fail. In this case the user must explicitly
## identify the 'y' variable via one of the two previous methods.
names( df )[which(names(df) == "y")] <- "response" # rename the 'y' variable to 'response'

get_y( df )  # now default behavior fails (i.e. returns NULL)

get_y( df, scoring_function_parameters = list( y_var = 'response' ) ) # this works
```

hazard_ratio *hazard_ratio*

Description

Computes the hazard ratio across treatment arms using a CoxPH model.

Usage

```
hazard_ratio(data, scoring_function_parameters = NULL)
```

Arguments

data	data.frame containing response data
scoring_function_parameters	named list of scoring function control parameters

Value

Hazard ratio across treatment arms.

See Also

TSDT, Surv, coxph,

mean_deviance_residuals *mean_deviance_residuals*

Description

Computes the mean of the deviance residuals from a survival model

Usage

```
mean_deviance_residuals(data, scoring_function_parameters = NULL)
```

Arguments

data	data.frame containing response data
scoring_function_parameters	named list of scoring function control parameters

Details

Computes the mean of the deviance residuals from a survival model. The deviance residual at time t is computed as the observed number of events at time t minus the expected number of events at time t (see Therneau, et. al. linked below). The expected number of events is the number of events predicted by the survival model. If the event under study is an undesirable event (as would likely be the case in a clinical context), then a smaller value for the deviance residual is desirable – i.e. it is desirable to observe fewer events than expected from the survival model. In this case the appropriate value for desirable_response in TSDT is desirable_response = 'decreasing'. If the event under study is desirable then the appropriate value for desirable_response is desirable_response = 'increasing'. It is assumed that most survival models are modeling an undesirable event. Therefore, when the user specifies mean_deviance_residual or diff_mean_deviance_residual, the default value for desirable_repsonse is changed to 'decreasing', unless the user explicitly provides desirable_response = 'increasing'. Note this differs from all other TSDT configurations, for which the default value for desirable_response is desirable_response = 'increasing'.

Value

Mean of deviance residuals

References

Therneau, T.M., Grambsch, P.M., and Fleming, T.R. (1990). Martingale-based residuals for survival models. Biometrika, 77(1), 147-160. doi:10.1093/biomet/77.1.147. https://academic.oup.com/biomet/article/77/1/147/271076

See Also

TSDT, Surv, coxph, survfit

`mean_response` *mean_response*

Description

Compute the mean response.

Usage

```
mean_response(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

This function will compute the mean of the response variable. If a value for trt_arm is provided the mean in that treatment arm only will be computed (and the trt variable must also be provided), otherwise the mean for all data passed to the function will be computed.

Value

The mean of the provided response variable.

See Also

TSDT, treatment_effect

Examples

```
N <- 50

data <- data.frame( continuous_response = numeric(N),
                    trt = character(N) )

data$continuous_response <- runif( min = 0, max = 20, n = N )
data$trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )

## Compute mean response for all data
mean_response( data, scoring_function_parameters = list( y_var = 'continuous_response' ) )
mean( data$continuous_response ) # Function return value should match this value

## Compute mean response for Experimental treatment arm only
scoring_function_parameters <- list( y_var = 'continuous_response', trt_arm = 'Experimental' )
mean_response( data, scoring_function_parameters = scoring_function_parameters )
# Function return value should match this value
mean( data$continuous_response[ data$trt == 'Experimental' ] )
```

MOB-class *MOB*

Description

MOB is a container class for trees created by mob.

Value

An object of class MOB

Slots

`tree` An object of class BinaryTree-class produced by mob.

`data` Training data.

`parameters` Control parameters

See Also

mob, BinaryTree-class

---

mob_wrapper *mob_wrapper*

Description

Wrapper function for mob.

Usage

```
mob_wrapper(
  response,
  x = NULL,
  z = NULL,
  covariates = NULL,
  tree_builder_parameters = list()
)
```

Arguments

response	Response variable to use in mob model.
x	Covariates passed to model in mob. mob uses fits the formula y ~ x1 + ... + xk | z1 + ... + zl where the variables before the | are passed to the model and the variables after the | are used for partitioning. x represents the x variables. See mob help page for more information.
z	Covariates used to parition the mob model. mob uses fits the formula y ~ x1 + ... + xk | z1 + ... + zl where the variables before the | are passed to the model and the variables after the | are used for partitioning. z represents the z variables. See mob help page for more information.
covariates	An alias for z.
tree_builder_parameters	A named list of parameters to pass to mob.

Value

An object of class MOB

See Also

mob

na2empty *na2empty*

Description

Replace all instances of NA in character variable with empty string.

Usage

```
na2empty(x)
```

Arguments

x	A character vector.

Value

A character vector with NA values replaced with empty string.

See Also

unfactor

Examples

```
## Create character variable with missing values
ex1 <- c( 'A', NA, 'B', NA, 'C', NA )
ex1

## Replace NAs with empty string
ex1 <- na2empty( ex1 )
ex1
```

parse_party *parse_party*

Description

Parse output from ctree() and mob() functions in party package.

Usage

```
parse_party(tree, data = NULL, include_subgroups = FALSE, digits = NULL)
```

Arguments

tree	An object of class BinaryTree or mob resulting from a call to the ctree() or mob() function.
data	data.frame containing covariates used to create tree.
include_subgroups	A logical value indicating whether or not to include a string representation of the subgroups in the results. Defaults to FALSE.
digits	Number of digits for rounding.

Details

Collects text output from party::ctree() or party::mob(), parses the splits, and populates a data.frame with the relevant data.

Value

A data.frame containing a parsed tree.

See Also

ctree, mob

Examples

```
requireNamespace( "party", quietly = TRUE )
requireNamespace( "modeltools", quietly = TRUE )
## From party::ctree() examples:
set.seed(290875)
## regression
airq <- subset(airquality, !is.na(Ozone))
airct <- party::ctree(Ozone ~ ., data = airq,
               controls = party::ctree_control(maxsurrogate = 3))

## Parse the results into a new data.frame
ex1 <- parse_party( airct )
ex1

## From party::mob() examples:
data("BostonHousing", package = "mlbench")
## and transform variables appropriately (for a linear regression)
BostonHousing$lstat <- log(BostonHousing$lstat)
BostonHousing$rm <- BostonHousing$rm^2
## as well as partitioning variables (for fluctuation testing)
BostonHousing$chas <- factor( BostonHousing$chas, levels = 0:1,
                              labels = c("no", "yes") )
BostonHousing$rad <- factor(BostonHousing$rad, ordered = TRUE)

## partition the linear regression model medv ~ lstat + rm
## with respect to all remaining variables:
fmBH <- party::mob( medv ~ lstat + rm | zn + indus + chas + nox + age +
```

```
            dis + rad + tax + crim + b + ptratio,
            control = party::mob_control(minsplit = 40), data = BostonHousing,
            model = modeltools::linearModel )

## Parse the results into a new data.frame
ex2 <- parse_party( fmBH )
ex2
```

`parse_rpart` *parse_rpart*

Description

Extract splits from an rpart.object returned from a call to rpart().

Usage

```
parse_rpart(tree, include_subgroups = FALSE)
```

Arguments

`tree`	An rpart.object returned from call to rpart().
`include_subgroups`	A logical value indicating whether or not to include a string representation of the subgroups in the results. Defaults to FALSE.

Details

This function takes as its input an rpart.object returned from a call to rpart. It parses this rpart.object using rpart_nodes() and returns the splits in the tree. The data returned include the NodeID of the node to split, the NodeID of that node's parent, the NodeID of that nodes left child and right child, the number of observations in that node, the variable used in the split, the data type for the splitting variable, the logic indicating which observations will go to the node's left child, the value of the splitting variable at which the split ocurrs, the mean response value of the node, and (optionally) the string representation of the node's subgroup. A node's subgroup is defined by the sequence of splits from the root to that node.

Value

A data.frame containing a parsed tree.

See Also

rpart_nodes, rpart, rpart.object

Examples

```
requireNamespace( "rpart", quietly = TRUE )
## Generate example data containing response, treatment, and covariates
N <- 50
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6),
               replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )

## Fit an rpart model
fit <- rpart::rpart( continuous_response ~ trt + X1 + X2 + X3 + X4,
                     control = rpart::rpart.control( maxdepth = 3L ) )
fit

## Parse the results into a new data.frame
ex1 <- parse_rpart( fit, include_subgroups = TRUE )
ex1
```

`partition` *partition*

Description

Partitions a vector x into n groups of roughly equal size.

Usage

```
partition(x, n)
```

Arguments

x	Vector to partition.
n	Number of (roughly) equally-sized groups

Value

A list of partitions of the vector x.

Examples

```
x <- 1:10
partition( x, 3 )
```

permutation *permutation*

Description

Permute response, treatment, or response for one treatment arm only.

Usage

```
permutation(response = NULL, trt = NULL, permute_arm = NULL)
```

Arguments

response	Response (or other) variable(s) to be permuted. This can be a data.frame of multiple variables (e.g. a data.frame of covariates or a multivariate response).
trt	Treatment variable.
permute_arm	reatment arm to permute.

Details

If a response variable is provided and treatment is not provided the response variable is permuted.

If a treatment variable is provided and response is not provided the treatment variable is permuted.

If a response variable and treatment variable and permute are provided the response variable is permuted only for the treatment arm indicated by permute_arm.

If a response variable and treatment variable are provided, but permute_arm

Value

If permuting response or treatment, returns vector of permuted response or treatment. If permuting response and treatment, returns a list of permuted response and treatment.

Examples

```
N <- 20
x <- data.frame( 1:N )
names( x ) <- "response"
x$trt <- factor( c( rep( "Experimental", 9 ), rep( "Control", N - 9 ) ) )
x$time <- x$response
x$event <- 0:1

## Permute treatment variable
ex1 <- x[,c("response","trt")]
ex1$permuted_trt <- permutation( trt = ex1$trt )

## Permute response variable
ex2 <- x[,c("response","trt")]
ex2$permuted_response <- permutation( response = ex2$response )
```

```
## Permute the response for treatment arm only
ex3 <- x[,c("response","trt")]
permuted3 <- permutation( response = ex3$response, trt = ex3$trt, permute_arm = "Experimental" )
names( permuted3 ) <- paste( "permuted_", names(permuted3), sep = "" )
ex3 <- cbind( ex3, permuted3 )

## Permute response and treatment together
ex4 <- x[,c("response","trt")]
permutation_list4 <- permutation( response = ex4$response, trt = ex4$trt )
ex4$permuted_response <- permutation_list4$response
ex4$permuted_trt <- permutation_list4$trt

## Permute a survival response for treatment arm only
ex5 <- x[,c("time","event","trt")]
permuted5 <- permutation( response = ex5[,c("time","event")], trt = ex5$trt,
                          permute_arm = "Experimental" )
names( permuted5 ) <- paste( "permuted_", names(permuted5), sep = "" )
ex5 <- cbind( ex5, permuted5 )

## Permute a survival outcome and treatment together
ex6 <- x[,c("time","event","trt")]
permutation_list6 <- permutation( response = ex6[,c("time","event")], trt = ex6$trt )
ex6$permuted_time <- permutation_list6$response$time
ex6$permuted_event <- permutation_list6$response$event
```

`quantile_response` *quantile_response*

Description

Return the specified quantile of the response distribution.

Usage

```
quantile_response(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

This function returns the response quantiles associated with a specified percentile. The default behavior is to return the median – i.e. 50th-percentile.

Value

A quantile of the response variable.

See Also

TSDT, diff_quantile_response, quantile

Examples

```
## Generate example data containing response and treatment
N <- 100
y = runif( min = 0, max = 20, n = N )
df <- as.data.frame( y )
names( df )  <- "y"
df$trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6),
                  replace = TRUE )

## Default behavior is to return the median
quantile_response( df )
median( df$y ) # should match previous result from quantile_response

## Get Q1 response
quantile_response( df, scoring_function_parameters = list( percentile = 0.25 ) )
quantile( df$y, 0.25 ) # should match previous result from quantile_response

## Get max response
quantile_response( df, scoring_function_parameters = list( percentile = 1 ) )
max( df$y ) # should match previous result from quantile_response
```

---

reset_factor_levels *reset_factor_levels*

Description

Reset the list of levels associated with a factor variable.

Usage

```
reset_factor_levels(data)
```

Arguments

data	A data.frame containing factor variables.

Details

After subsetting a factor variable some factor levels that were previously present may be lost. This is particularly true for relatively rare factor levels. This function resets the list of factor levels to include only the levels currently present.

Value

A data.frame with factor variable that now have reset levels.

Examples

```
ex1 = as.factor( c( rep('A', 3), rep('B',3), rep('C',3) ) )

## The levels associated with the factor variable include the letters A, B, C
ex1  # Levels are A, B, C

## If the last three observations are dropped the value C no longer occurs
## in the data, but the list of associated factor levels still contains C.
## This mismatch between the data and the list of factor levels may cause
## problems, particularly for algorithms that iterate over the factor levels.

ex1 <- ex1[1:6]
ex1 # Levels are still A, B, C, but the data contains only A and B

## If the factor levels are reset the data and list of levels will once again
## be consistent
ex1 <- reset_factor_levels( ex1 )
ex1 # Levels now contain only A and B, which is consistent with data
```

---

rpart_nodes *rpart_nodes*

Description

Extract node information from an rpart.object.

Usage

```
rpart_nodes(tree)
```

Arguments

tree	An rpart.object returned from call to rpart().

Details

Information about nodes and splits returned in an rpart.object is contained in strings printed to the console. This function parses those strings and populates a data.frame.

Value

A data.frame containing the nodes of a parsed tree.

See Also

rpart, rpart.object

Examples

```
requireNamespace( "rpart", quietly = TRUE )
## Generate example data containing response, treatment, and covariates
N <- 50
continuous_response = runif( min = 0, max = 20, n = N )
binary_response <- sample( c('A','B'), size = N, prob = c(0.5,0.5),
                           replace = TRUE )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6),
               replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )

## Fit an rpart model with continuous response (i.e. regression)
fit1 <- rpart::rpart( continuous_response ~ trt + X1 + X2 + X3 + X4 )
fit1

## Parse the results into a new data.frame
ex1 <- rpart_nodes( fit1 )
ex1

## Fit an rpart model with binary response (i.e. classification)
fit2 <- rpart::rpart( binary_response ~ trt + X1 + X2 + X3 + X4 )
fit2
```

---

rpart_wrapper *rpart_wrapper*

Description

A wrapper function to rpart.

Usage

```
rpart_wrapper(
  response,
  response_type = NULL,
  covariates = NULL,
  tree_builder_parameters = NULL,
  prune = FALSE
)
```

Arguments

response	Response variable to use in rpart model.
response_type	Class of response variable.
covariates	Covariates to use in rpart model.

tree_builder_parameters
: A named list of parameters to pass to rpart. This includes all input parameters that rpart can take.

prune
: Logical variable indicating whether the tree shold be pruned to the subtree with the smallest cross-validation error. Defaults to FALSE.

Details

This function provides a wrapper to rpart that provides a convenient interface for specifying the response variable and covariates for the rpart model. The user may indicate whether the tree should be pruned to the size that yields the smallest cross-validation error. An rpart.object is returned.

Value

An object of class rpart.

See Also

rpart, rpart.object, Surv

Examples

```
## Generate example data containing response, treatment, and covariates
N <- 100
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )
covariates <- data.frame( trt )
names( covariates ) <- "trt"
covariates$X1 <- X1
covariates$X2 <- X2
covariates$X3 <- X3
covariates$X4 <- X4
## Fit an rpart model
ex1 <- rpart_wrapper( response = continuous_response, covariates = covariates )
ex1
```

---

subgroup *subgroup*

Description

Subset a user-provided data.frame according to the subgroup specified by a node in a tree.

Usage

```
subgroup(splits, node, xdata, ydata = xdata)
```

Arguments

splits	A data.frame of splits returned from a call to parse_rpart().
node	The NodeID of the node defining the desired split.
xdata	The data.frame of covariates to subset according to the subgroup definition.
ydata	The associated vector of response values to subset according to the subgroup definition. (optional)

Details

After the splits from an rpart.object are extracted by a call to parse_rpart(), the extracted splits define a subgroup for each node. This subgroup can be used to subset a user-provided data.frame. This function takes as its input a data.frame of splits obtained from a call to parse_rpart(), a NodeID indicating which node specifies the desired subgroup, a data.frame of covariates to subset, and (optionally) the associated response data to subset. If only xdata is specified by the user, the subset of xdata implied by the subgroup will be returned. If xdata and ydata are provided by the user, the subset of ydata will be returned (xdata is still required from the user because the subsetting is computed on the covariate values even when the data returned to the user are from ydata).

Value

A data.frame containing the data consistent with the specified subgroup.

See Also

parse_rpart, rpart, rpart.object

Examples

```
requireNamespace( "rpart", quietly = TRUE )

## Generate example data containing response, treatment, and covariates
N <- 20
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )

covariates <- data.frame( trt )
names( covariates ) <- "trt"
covariates$X1 <- X1
covariates$X2 <- X2
covariates$X3 <- X3
covariates$X4 <- X4

## Fit an rpart model
fit <- rpart::rpart( continuous_response ~ trt + X1 + X2 + X3 + X4 )

## Return parsed splits with subgroups
```

```
splits1 <- parse_rpart( fit, include_subgroups = TRUE )
splits1

## Subset covariate data according to split for NodeID 3
ex1 <- subgroup( splits = splits1, node = 3, xdata = covariates )
ex1

## Subset response data according to split for NodeID 3
ex2 <- subgroup( splits = splits1, node = 3, xdata = covariates, ydata = continuous_response )
ex2
```

---

subsample *subsample*

Description

Generate a vector of subsamples.

Usage

```
subsample(
  x,
  trt = NULL,
  trt_control = "Control",
  training_fraction = NULL,
  validation_fraction = NULL,
  test_fraction = NULL,
  n_samples = 1
)
```

Arguments

x	<Source data to subsample.
trt	Treatment variable. (optional)
trt_control	Value for treatment control arm. Defaulte value is 'Control'.
training_fraction	Fraction of source data to include in training subsample.
validation_fraction	Fraction of source data to include in validation subsample.
test_fraction	Fraction of source data to include in test subsample.
n_samples	Number of subsamples to generate.

Details

Each subsample will contain training, validation, and test data in proportions specified by the user. If a treatment variable is supplied the ratio of treatments will be preserved as closely as possible.

Value

Vector of objects of class Subsample.

See Also

Subsample

Examples

```
## Generate example data frame containing response and treatment
N <- 50
x <- data.frame( runif( N ) )
names( x ) <- "response"
x$treatment <- factor( sample( c("Control","Experimental"), size = N,
                       prob = c(0.8,0.2), replace = TRUE ) )

## Generate two subsamples
ex1 <- subsample( x,
                  training_fraction = 0.9,
                  test_fraction = 0.1,
                  n_samples = 2 )

## Generate two subsamples preserving treatment ratio
ex2 <- subsample( x,
                  trt = x$treatment,
                  trt_control = "Control",
                  training_fraction = 0.7,
                  validation_fraction = 0.2,
                  test_fraction = 0.1,
                  n_samples = 2 )
```

Subsample-class *Subsample*

Description

Subsmaple is a container class for subsamples.

Value

Object of class

Slots

`training` Training data.

`validation` Validation data.

`test` Test data.

See Also

subsample

`summary,TSDT-method` *Summary function for class TSDT.*

Description

Summary function for class TSDT.

Usage

```
## S4 method for signature 'TSDT'
summary(object)
```

Arguments

`object`	An object of class TSDT.

Value

A data.frame containing the superior subgroups identified by TSDT.

See Also

TSDT

`survival_time_quantile` *survival_time_quantile*

Description

Computes the quantile of a survival function.

Usage

```
survival_time_quantile(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

Computes the quantile of a survival function. The user specifies the percentile associated with the desired quantile in scoring_function_parameters. The default is percentile = 0.50, which returns the median survival. A user may also specify a value for the trt_arm parameter in scoring_function_parameters to compute the survival quantile for only one arm.

Value

A quantile of the response survival time.

See Also

TSDT, diff_survival_time_quantile, Surv, coxph, survfit, survreg, quantile.survfit, predict.coxph, predict.survreg

Examples

```
N <- 200
time <- runif( min = 0, max = 20, n = N )
event <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )
df <- data.frame( y = survival::Surv( time, event ), trt = trt )

## Compute median survival time in Experimental treatment arm.
ex1 <- survival_time_quantile( data = df,
                               scoring_function_parameters = list( trt_var = "trt",
                               trt_arm = "Experimental",
                               percentile = 0.50 ) )

## Compute Q1 survival time for all data. It is necessary here to explicitly
## specify trt = NULL because a variable called trt exists in df. The default
## behavior is to use this variable as the treatment variable. To override
## the default behavior trt = NULL is included in scoring_function_parameters.
ex2 <- survival_time_quantile( data = df,
                     scoring_function_parameters = list( trt = NULL, percentile = 0.25 ) )
```

---

treatment_effect *treatment_effect*

Description

Compute treatment effect as mean(treatment response) - mean(control response)

Usage

```
treatment_effect(data, scoring_function_parameters = NULL)
```

Arguments

`data`	data.frame containing response data
`scoring_function_parameters`	named list of scoring function control parameters

Details

This function will compute the treatment for the response. The treatment effect is computed as the difference in means between the non-control treatment arm and the control treatment arm. The user must provide the treatment variable as well as the control value.

Value

The difference in mean response across treatment arms.

See Also

TSDT, mean_response

Examples

```
N <- 100

df <- data.frame( continuous_response = numeric(N),
                  trt = integer(N) )

df$continuous_response <- runif( min = 0, max = 20, n = N )
df$trt <- sample( c(0,1), size = N, prob = c(0.4,0.6), replace = TRUE )

# Compute the treatment effect
treatment_effect( df, list( y_var = 'continuous_response', trt_control = 0 ) )

# Function return value should match this value
mean( df$continuous_response[df$trt == 1] ) - mean( df$continuous_response[df$trt == 0] )
```

TSDT *Treatment-Specific Subgroup Detection Tool*

Description

Implements a method for identifying subgroups with superior response relative to the overall sample.

Usage

```
TSDT(
  response = NULL,
  response_type = NULL,
  survival_model = "kaplan-meier",
  percentile = 0.5,
  tree_builder = "rpart",
  tree_builder_parameters = list(),
  covariates,
  trt = NULL,
  trt_control = 0,
  permute_method = NULL,
  permute_arm = NULL,
  n_samples = 1,
  desirable_response = NULL,
  sampling_method = "bootstrap",
  inbag_proportion = 0.5,
  scoring_function = NULL,
  scoring_function_parameters = list(),
  inbag_score_margin = 0,
  oob_score_margin = 0,
  eps = 1e-05,
  min_subgroup_n_control = NULL,
  min_subgroup_n_trt = NULL,
  min_subgroup_n_oob_control = NULL,
  min_subgroup_n_oob_trt = NULL,
  maxdepth = .Machine$integer.max,
  rootcompete = 0,
  competedepth = 1,
  strength_cutpoints = c(0.1, 0.2, 0.3),
  n_permutations = 0,
  n_cpu = 1,
  trace = FALSE
)
```

Arguments

response	Response variable.
response_type	Data type of response. Must be one of binary, continuous, survival. If none provided it will be inferred from the data type of response. (optional)
survival_model	The model to use for a survival response. Defaults to kaplan-meier. Other possible values are: coxph, fleming-harrington, fh2, weibull, exponential, gaussian, logistic, lognormal, and loglogistic. (optional)
percentile	For a two-arm study this parameter specifies a test for the difference in response percentile across the two treatment arms. For a continuous response the default value for percentile is NULL. Instead, the difference in mean response is computed by default for a continuous response. If the user provides a values of

percentile = 0.50 then the difference in median response would be computed. For a survival outcome, the default value for percentile is 0.50, which computes the difference in median survival.

`tree_builder` The algorithm to use for building the trees. Defaults to rpart. Other possible values include ctree and mob (both from the party package). (optional)

`tree_builder_parameters`
A named list of parameters to pass to the tree-builder. The default tree-builder is rpart. In this case, the parameters passed here would be rpart parameters. Examples might include parameters such as control, cost, weights, na.action, etc. Consult the rpart documentation (or the documentation of your selected tree-builder) for a complete list. (optional)

`covariates` A data.frame containing the covariates.

`trt` Treatment variable. Only needed if there are two treatment arms. (optional)

`trt_control` Value for treatment control arm. This parameter is relevant only for two-arm data. (defaults to 0)

`permute_method` Indicates whether only the response variable should be permuted in the computation of the p-value, or the response and treatment variable should be permuted together (preserving the treatment-response correlation, but eliminating the correlation with the covariates), or the response variable should be permuted within one treatment arm only. The parameter values for these permutation schemes are (respectively) simple, permute_response_and_treatment, and permute_response_one_arm. See permute_arm to specify which treatment arm is to be permuted. The default permutation scheme is response_one_arm. As noted in the documentation for the permute_arm parameter is to permute the non-control arm. Taken together, this implies the default permutation method for p-value computation is to permute the response in the non-control arm only. For one-arm data only the response is permuted. (optional)

`permute_arm` Which treatment arm should be permuted? Defaults to the experimental treatment arm – i.e. the treatment arm not matching the value provided in trt_control. For one-arm data only the response is permuted. (optional)

`n_samples` Number of TSDT_Samples to draw.

`desirable_response`
Direction of desirable response. Valid values are 'increasing' or 'decreasing'. The default value is 'increasing'. It is important to note that although the parameter is called desirable_response, it actually refers to the desirable direction of scoring function values. In most cases there is a positive correlation bewteen the response and scoring function values – i.e. as the response increases the scoring function also increases. One instance for which this relationship between response and scoring function may not hold is when mean_deviance_residuals or diff_mean_deviance_residuals is used as the scoring function. See the help for these scorings function for further details.

`sampling_method`
Sampling method used to populate samples for TSDT in-bag and out-of-bag data. Must be either bootstrap or subsample. Default is bootstrap.

`inbag_proportion`
The proportion of the data to use as the in-bag subset when sampling_method is subsample.

`scoring_function`
: Scoring function to compute treatment effect. Links to several possible scoring functions are provided in the See Also section below.

`scoring_function_parameters`
: Parameters passed to the scoring function. As an example, the scoring function quantile_response takes a parameter "percentile" which indicates the desired percentile of the response distribution. Thus, if the median response is desired, this parameter could be set as follows: scoring_function_parameters = list(percentile = 0.50). Most of the built-in scoring functions have sensible defaults for the scoring function parameters so it is not necessary to specify them explicitly in the call to TSDT. But this parameter could be very useful for user-defined custom scoring functions. (optional)

`inbag_score_margin`
: Required margin above overall mean for a subgroup to be considered superior. If a subgroup mean must be 10% larger than the overall subgroup mean to be superior then inbag_score_margin = 0.10. If desirable_response = "decreasing" then inbag_score_margin should be negative or zero.

`oob_score_margin`
: Similar to inbag_score_margin but for classifying out-of-bag subgroups as superior.

`eps`
: Tolerance value for floating-point precision. The default is 1E-5. (optional)

`min_subgroup_n_control`
: Minimum number of Control arm observations in an in-bag subgroup. A value greater than or equal to one will be interpreted as the required minimum number of observations. A value between zero and one will be interpreted as a proportion of the in-bag Control observations. For a bootstrapped in-bag sample the default for this parameter is 10 of Control observations in the overall sample. For an in-bag sample obtained via subsampling the default value is the inbag_proportion times 10 number of Control observations in the overall sample.

`min_subgroup_n_trt`
: Minimum number of Experimental arm observations in an in-bag subgroup. A value greater than or equal to one will be interpreted as the required minimum number of observations. A value between zero and one will be interpreted as a proportion of the in-bag Experimental observations. For a bootstrapped in-bag sample the default for this parameter is 10 number of Experimental observations in the overall sample. For an in-bag sample obtained via subsampling the default value is the inbag_proportion times 10% of the number of Experimental observations in the overall sample.

`min_subgroup_n_oob_control`
: Minimum number of Control arm observations in an out-of-bag subgroup. A value greater than or equal to one will be interpreted as the required minimum number of observations. A value between zero and one will be interpreted as a proportion of the out-of-bag Control observations. For a bootstrapped out-of-bag sample the default for this parameter is exp(-1)*10% of the number of Control observations in the overall sample. For an out-of-bag sample obtained via subsampling the default value is the inbag_proportion times (1-inbag_proportion)*10 Control observations in the overall sample.

`min_subgroup_n_oob_trt`
: Minimum number of Experimental arm observations in an out-of-bag subgroup. A value greater than or equal to one will be interpreted as the required minimum number of observations. A value between zero and one will be interpreted as a proportion of the out-of-bag Experimental observations. For a bootstrapped out-of-bag sample the default for this parameter is exp(-1)*10% of the number of Experimental observations in the overall sample. For an out-of-bag sample obtained via subsampling the default value is the inbag_proportion times (1-inbag_proportion)*10% of the number of Experimental observations in the overall sample.

`maxdepth`
: Maximum depth of trees.

`rootcompete`
: Number of competitor splits to retain for root node split.

`competedepth`
: Depth of competitor split trees (defaults to 1)

`strength_cutpoints`
: Cutpoints for permuted p-values to classify a subgroup as Strong, Moderate, Weak, or Not Confirmed. The default cutpoints are 0.10, 0.20, and 0.30 for Strong, Moderate, and Weak subgroups, respectively. (optional)

`n_permutations`
: Number of permutations to compute for adjusted p-value. Defaults to zero (no p-value computation). If p-values are desired, it is recommended to use at least 500 permutations.

`n_cpu`
: Number of CPUs to use. Defaults to 1.

`trace`
: Report number of permutations computed as algorithm proceeds.

Details

The Treatment-Specific Subgroup Detection Tool (TSDT) creates several bootstrapped samples from the input data. For each of these bootstrapped samples the in-bag and out-of-bag data are retained. A tree is grown on the in-bag data of each bootstrapped sample using the response variable and supplied covariates. Each split in the tree defines a subgroup. The overall mean response for the in-bag data is computed as well as the mean response within each subgroup. Additionally, a scoring function is provided. Example scoring functions might be mean response, difference in mean response between treatment arms (i.e. treatment effect), or a quantile of the response (e.g. median), or a difference in quantiles across treatment arms. Sensible defaults are provided given the data type of the response and treatment variables. The user can also specify a custom scoring function. The value of the scoring function is computed for the overall in-bag data and each subgroup. Subgroups with mean response larger than the overall in-bag mean response and a mean scoring function value larger than the overall in-bag scoring function value are identified as superior subgroups. This definition of a superior subgroup assumes a larger value of the response variable is desirable. If a smaller value of the response is desirable then subgroups with mean response and mean scoring function smaller than the overall in-bag mean are superior. The same computation of overall and subgroup mean response and mean scoring function are done for the out-of-bag data. This is repeated for all bootstrapped samples. Measures of internal and external consistency are then computed. Internal consistency is computed for each subgroup that is identified as superior in one of the in-bag samples. Internal consistency for each of these subgroups is the fraction of bootstrapped samples where that subgroup is identified as superior in the in-bag data. External consistency is also defined only for subgroups that are identified as superior in at least one of the in-bag samples. For each of these subgroups, external consistency is the number of bootstrapped

samples where the subgroup is defined as superior in the in-bag and out-of-bag data divided by the number of bootstrapped samples where the subgroup is identified as superior in the in-bag data. The internal and external consistency results are returned for each subgroup that identified as superior in the in-bag data of at least one bootstrapped sample. A score for the overall strength of each subgroup is computed as the product of the internal and external consistency. Optionally, a permutation-adjusted p-value for the strength of each subgroup can be computed. Based on this p-value subgroups are classified as strong, moderate, weak, or not confirmed. A suggested cutoff for each subgroup is also provided. This is helpful because two subgroups defined on the same continuous splitting variable but with different cutpoints are considered equivalent. That is, one subgroup X1<0.6 and another X1<0.7 would be considered equivalent and listed in the results as X1<xxxxx. (Note that X1<0.6 and X1>=0.7 would be considered distinct subgroups and listed in the output as X1<xxxxx and X1>=xxxxx, respectively.) So if a subgroup listed in the output as X1<xxxxx could actually represent many different numeric values for xxxxx it is helpful to provide a final suggestion for the cutpoint. The algorithm retains all the numeric values and uses the median as the suggested cutoff. The user can also request the vector of numeric cutpoints and use any function of their choosing to compute a suggested cutoff.

Value

An object of class TSDT

Author(s)

Brian Denton <denton_brian_david@lilly.com>, Chakib Battioui <battioui_chakib@lilly.com>, Lei Shen <shen_lei@lilly.com>

References

Battioui, C., Shen, L., Ruberg, S., (2014). A Resampling-based Ensemble Tree Method to Identify Patient Subgroups with Enhanced Treatment Effect. JSM proceedings, 2014

Shen, L., Battioui, C., Ding, Y., (2013). Chapter "A Framework of Statistical methods for Identification of Subgroups with Differential Treatment Effects in Randomized Trials" in the book "Applied Statistics in Biomedicine and Clinical Trials Design"

See Also

mean_response, quantile_response, diff_quantile_response, treatment_effect, survival_time_quantile, diff_survival_time_quantile, mean_deviance_residuals, diff_mean_deviance_residuals, diff_restricted_mean_survival_time, TSDT, rpart, ctree, mob

Examples

```
## Create example data for constructing TSDT object
N <- 200
continuous_response = runif( min = 0, max = 20, n = N )
trt <- sample( c('Control','Experimental'), size = N, prob = c(0.4,0.6), replace = TRUE )
X1 <- runif( N, min = 0, max = 1 )
X2 <- runif( N, min = 0, max = 1 )
X3 <- sample( c(0,1), size = N, prob = c(0.2,0.8), replace = TRUE )
X4 <- sample( c('A','B','C'), size = N, prob = c(0.6,0.3,0.1), replace = TRUE )
```

```
covariates <- data.frame( X1 )
covariates$X2 <- X2
covariates$X3 <- factor( X3 )
covariates$X4 <- factor( X4 )


## In the following examples n_samples and n_permutations are set to small
## values so the examples complete quickly. The intent here is to provide
## a small functional example to demonstrate the structure of the output. In
## a real-world use of TSDT these values should be at least 100 and 500,
## respectively.

## Single-arm TSDT
ex1 <- TSDT( response = continuous_response,
            covariates = covariates[,1:4],
            inbag_score_margin = 0,
            desirable_response = "increasing",
            n_samples = 5,       ## use value >= 100 in real world application
            n_permutations = 5,  ## use value >= 500 in real world application
            rootcompete = 1,
            maxdepth = 2 )

## Two-arm TSDT
ex2 <- TSDT( response = continuous_response,
            trt = trt, trt_control = 'Control',
            covariates = covariates[,1:4],
            inbag_score_margin = 0,
            desirable_response = "increasing",
            oob_score_margin = 0,
            min_subgroup_n_control = 10,
            min_subgroup_n_trt = 20,
            maxdepth = 2,
            rootcompete = 1,
            n_samples = 5,      ## use value >= 100 in real world application
            n_permutations = 5 ) ## use value >= 500 in real world application
```

`TSDT-class` *TSDT*

Description

TSDT is a container class for TSDT samples and metadata.

Value

Object of class TSDT

Slots

`parameters` List of parameters used in construction of TSDT samples.

`samples` Vector of TSDT_Sample objects.

`superior_subgroups` data.frame containing summary statistics for superior subgroups

`cutpoints` An object of class TSDT_CutpointDistribution.

`distributions` A list of distributions of TSDT statistics.

See Also

TSDT, TSDT_Sample, TSDT_CutpointDistribution

`TSDT_CutpointDistribution-class` *TSDT_CutpointDistribution*

Description

Implementation of TSDT_CutpointDistribution class. This class continuous split variable. If the subgroup contains more than one split variable a distribution of numeric cutpoints is collected for each continuous split variable in the subgroup definition.

Value

Object of class TSDT_CutpointDistribution

Slots

`Cutpoints` An object of class hash-class

See Also

TSDT, hash

`TSDT_Sample-class` *TSDT_Sample*

Description

TSDT_Sample is a container class containing the in-bag and out-of-bag data from a subsampled or bootstrapped dataset. This container class also contains a data.frame containing the parsed tree that is fit on the in-bag data.

Value

Object of class TSDT_Sample

Slots

`inbag` A data.frame containing in-bag data

`oob` A data.frame containing out-of-bag data

`subgroups` A data.frame containing a parsed tree

See Also

TSDT

`unfactor` *unfactor*

Description

Convert the factor columns of a data.frame to character or numeric.

Usage

```
unfactor(data)
```

Arguments

`data`	A factor variable or a data.frame containing factor variables.

Details

If the levels of a factor variable in data represent numeric values the variable will be converted to a numeric data type, otherwise it is converted to a character data type.

Value

A vector or data.frame no longer containing any factor variables.

See Also

na2empty

Examples

```
## Generate example data.frame of factors with factor levels of numeric,
## character and mixed data types.
N <- 20
ex1 <- data.frame( factor( sample( c(0,1,NA), size = N, prob = c(0.4,0.3,0.3),
                           replace = TRUE ) )  )
names( ex1 ) <- "num"
ex1$char <- factor( sample( c("Control","Experimental", NA ), size = N,
                    prob = c(0.4,0.3,0.3), replace = TRUE ) )
ex1$mixed <- factor( sample( c(10,'A',NA), size = N, prob = c(0.4,0.3,0.3),
```

```
                    replace = TRUE ) )

## Initially the data type of all variables in ex1 is factor
ex1
class( ex1$num )   #factor
class( ex1$char )  #factor
class( ex1$mixed ) #factor

## Now convert all factor variables to numeric or character
ex2 <- unfactor( ex1 )
ex2

## The data types are now numeric or character
class( ex2$num )   # numeric
class( ex2$char )  # character
class( ex2$mixed ) # character

## The <NA> notation for missing factor values that have been converted to
## character can be changed to an empty string for easier reading by use of
## the function na2empty().
ex2$char <- na2empty( ex2$char )
ex2$mixed <- na2empty( ex2$mixed )
ex2
```

---

unpack_args *unpack_args*

Description

Assign the elements of a named list in current environment.

Usage

```
unpack_args(args)
```

Arguments

args	List of entities to be assigned.

Details

This function takes a list of named entities and assigns each element of the list to its name in the calling environment.

See Also

assign, parent.frame

Examples

```
## Create a list of named elements
arglist <- list( one = 1, two = 2, color = "blue" )

## The variables one, two, and color do not exist in the current environment
ls()

## Unpack the elements in arglist
unpack_args( arglist )

## Now the variables one, two, and color do exist in the current environment
ls()
one
```

`%nin%` *%nin%*

Description

Negation of the built-in %in% operator. %nin% is a short-hand for !(a %in% b).

Usage

```
a %nin% b
```

Arguments

a	Any R object for which the binary operator %in% is defined. This would include many built-in R primitives.
b	Any R object for which the binary operator %in% is defined. This would include many built-in R primitives.

See Also

%in%

Examples

```
# 4 is not an element in {5,6,7}.
4 %nin% 5:7  # Evaluates to TRUE

# 4 is an element in {4,5,6,7}.
4 %nin% 4:7  # Evaluates to FALSE
```

Index