Package 'riAFTBART'

July 23, 2025

Type Package

Title A Flexible Approach for Causal Inference with Multiple Treatments and Clustered Survival Outcomes

Version 0.3.3

Description Random-intercept accelerated failure time (AFT) model utilizing Bayesian additive regression trees (BART) for drawing causal inferences about multiple treatments while accounting for the multilevel survival data structure. It also includes an interpretable sensitivity analysis approach to evaluate how the drawn causal conclusions might be altered in response to the potential magnitude of departure from the no unmeasured confounding assumption.This package implements the methods described by Hu et al. (2022) <doi:10.1002/sim.9548>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Imports MCMCpack, msm, dbarts, magrittr, foreach, doParallel, dplyr, BART, stringr, tidyr, survival, cowplot, ggplot2, twang, nnet, RRF, randomForest

NeedsCompilation no

Author Liangyuan Hu [aut], Jiayi Ji [aut], Fengrui Zhang [cre]

Maintainer Fengrui Zhang <fz174@sph.rutgers.edu>

Repository CRAN

Date/Publication 2024-05-29 23:20:09 UTC

Contents

`cal_PEHE` *Calculate the PEHE*

Description

This function calculates the PEHE based on the survival probability from a fitted ri-AFTBART model.

Usage

```
cal_PEHE(object, metric, time, LP, lambda, eta)
```

Arguments

`object`	An object from cal_survprob() function.
`metric`	A character string representing the metric to be calculated for PEHE. Only `"survival"` is allowed.
`time`	A numeric value representing the time point used to calculate PEHE.
`LP`	A numeric vector corresponding to the true linear predictors for each treatment from the simulated data.
`lambda`	A numeric value representing the true follow up time for from the simulated data.
`eta`	A numeric value to induce proportional/non-proportional hazards assumption from the simulated data.

Value

A list with the following three components:

`true:`	A numeric vector representing the true survival or rmst for each individual.
`predicted:`	A numeric vector representing the predicted survival or rmst for each individual.
`pehe:`	A numeric vector representing the calculated pehe.

Examples

```
library(riAFTBART)
lp_w_all <-
  c(".4*x1 + .1*x2  - .1*x4 + .1*x5",    #' w = 1
    ".2 * x1 + .2 * x2  - .2 * x4 - .3 * x5")  #' w = 2
nlp_w_all <-
  c("-.5*x1*x4  - .1*x2*x5", #' w = 1
    "-.3*x1*x4 + .2*x2*x5")#' w = 2
lp_y_all <- rep(".2*x1 + .3*x2 - .1*x3 - .1*x4 - .2*x5", 3)
nlp_y_all <- rep(".7*x1*x1  - .1*x2*x3", 3)
X_all <- c(
  "rnorm(10, 0, 0.5)",#' x1
  "rbeta(10, 2, .4)",   #' x2
  "runif(10, 0, 0.5)",#' x3
  "rweibull(10,1,2)",  #' x4
  "rbinom(10, 1, .4)"#' x5
)
set.seed(111111)
data <- dat_sim(
  nK = 2,
  K = 5,
  n_trt = 3,
  X = X_all,
  eta = 2,
  lp_y = lp_y_all,
  nlp_y  = nlp_y_all,
  align = FALSE,
  lp_w = lp_w_all,
  nlp_w = nlp_w_all,
  lambda = c(1000,2000,3000),
  delta = c(0.5,0.5),
  psi = 1,
  sigma_w = 1,
  sigma_y = 2,
  censor_rate = 0.1
)
data$LP_true[,1]
data$lambda
data$eta
res <- riAFTBART_fit(M.burnin = 10, M.keep = 10, M.thin = 1, status = data$delta,
                      y.train = data$Tobs, trt.train = data$w, trt.test = 1,
                      x.train = data$covariates,
                      x.test = data$covariates,
                      cluster.id = data$cluster)
res_cal_surv_prob <- cal_surv_prob(object = res,
time.points = 1:max(data$Tobs),
test.only = TRUE,
cluster.id = data$cluster)

res_cal_PEHE_survival <- cal_PEHE(object = res_cal_surv_prob,
                         metric = "survival", time = 40,
                         LP = data$LP_true[,1], lambda = data$lambda[1],
                         eta = data$eta)

res_cal_PEHE_rmst <- cal_PEHE(object = res_cal_surv_prob,
                                  metric = "rmst",
                                  time = 40,
                                  LP = data$LP_true[,1],
                                  lambda = data$lambda[1],
                                  eta = data$eta)
```

`cal_surv_prob` *Calculate the survival probability from a fitted riAFT-BART model*

Description

This function calculates the individual survival probability from a fitted riAFT-BART model at desired values of times

Usage

```
cal_surv_prob(
  object,
  time.points,
  test.only = FALSE,
  train.only = FALSE,
  cluster.id
)
```

Arguments

`object`	A fitted object from riAFTBART_estimate() function.
`time.points`	A numeric vector representing the points at which the survival probability is computed.
`test.only`	A logical indicating whether or not only data from the test set should be computed. The default is FALSE.
`train.only`	A logical indicating whether or not only data from the training set should be computed. The default is FALSE.
`cluster.id`	A vector of integers representing the cluster id. The cluster id should be an integer and start from 1.

Value

A list with the following two components

`Surv:`	A matrix of survival probabilities for each individual.
`time.points:`	The time point entered.

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 50      # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = stats::rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
Y = pmin(y,C)
status = as.numeric(y<=C)
res <- riAFTBART_fit(M.burnin = 50, M.keep = 50, M.thin = 1, status = status,
                      y.train = Y, trt.train = trt.train, trt.test = trt.test,
                      x.train = cbind(x1,x2),
                      x.test = cbind(x1,x2),
                      cluster.id = cluster.id)

surv_prob_res <- cal_surv_prob(object = res, time.points = sort(exp(logtime)),
test.only = TRUE, cluster.id = cluster.id)
```

`dat_sim` *Simulate data with multiple treatments and clustered survival outcomes*

Description

This function simulate data with multiple treatments and clustered survival outcomes. Users can adjust the following 11 design factors: (1) The number of clusters, (2) the sample size in each cluster, (3) ratio of units across treatment groups, (4) whether the treatment assignment model and the outcome generating model are linear or nonlinear, (5) whether the covariates that best predict the treatment also predict the outcome well, (6) whether the response surfaces are parallel across treatment groups, (7) degree of covariate overlap, (8) Whether the proportional hazards assumption is satisfied, (9) mean follow up time for each treatment group, (10) censoring proportion and (11) Standard deviation for the cluster effect in the treatment assignment and outcome generating model.

Usage

```
dat_sim(
  nK,
  K,
  n_trt,
  X,
  lp_y,
  nlp_y,
  align = TRUE,
  eta,
  lambda,
  delta,
  psi,
  lp_w,
  nlp_w,
  sigma_w,
  sigma_y,
  censor_rate
)
```

Arguments

`nK`	A numeric value indicating the number of clusters.
`K`	A numeric value indicating the sample size in each cluster.
`n_trt`	A numeric value indicating the number of treatments.
`X`	A vector of characters representing covariates, with each covariate being generated from the standard probability `distributions` in the `stats` package.
`lp_y`	A vector of characters of length `n_trt`, representing the linear effects in the outcome generating model.
`nlp_y`	A vector of characters of length `n_trt`, representing the nonlinear effects in the outcome generating model.
`align`	A logical indicating whether the predictors in the treatment assignment model are the same as the predictors for the outcome generating model. The default is `TRUE`. If the argument is set to `FALSE`, users need to specify additional two arguments `lp_w` and `nlp_w`.
`eta`	A numeric value to induce proportional hazards assumption or a character including linear combination of Xs to induce nonproportional hazards assumption.
`lambda`	A numeric vector of length `n_trt` inducing different follow up time across treatment groups.
`delta`	A numeric vector of length `n_trt`-1 inducing different ratio of units across treatment groups.
`psi`	A numeric value for the parameter governing the sparsity of covariate overlap.
`lp_w`	A vector of characters of length `n_trt` - 1, representing the treatment assignment model.

`nlp_w`	A vector of characters of length `n_trt` - 1, representing the treatment assignment model.
`sigma_w`	A numeric value representing the standard deviation for the cluster effect in the treatment assignment model.
`sigma_y`	A numeric value representing the standard deviation for the cluster effect in the outcome generating model.
`censor_rate`	A numeric value for the rate parameter governing the proportion of censoring.

Value

A list with 7 elements for simulated data. It contains

`covariates:`	X matrix
`w:`	treatment indicators
`Tobs:`	observed follow up time for the simulated right censored data
`status:`	the censoring indicator
`cluster:`	the clustering indicator
`censor_prop:`	the censoring proportion
`T_mean:`	mean observed follow up time
`ratio_of_units:`	the proportions of units in each treatment group

Examples

```
library(riAFTBART)
lp_w_all <-
  c(".4*x1 + .1*x2  - .1*x4 + .1*x5",    # w = 1
    ".2 * x1 + .2 * x2  - .2 * x4 - .3 * x5")  # w = 2
nlp_w_all <-
  c("-.5*x1*x4  - .1*x2*x5", # w = 1
    "-.3*x1*x4 + .2*x2*x5")# w = 2
lp_y_all <- rep(".2*x1 + .3*x2 - .1*x3 - .1*x4 - .2*x5", 3)
nlp_y_all <- rep(".7*x1*x1  - .1*x2*x3", 3)
X_all <- c(
  "rnorm(1000, 0, 0.5)",# x1
  "rbeta(1000, 2, .4)",   # x2
  "runif(1000, 0, 0.5)",# x3
  "rweibull(1000,1,2)",  # x4
  "rbinom(1000, 1, .4)"# x5
)
set.seed(111111)
data <- dat_sim(
  nK = 20,
  K = 50,
  n_trt = 3,
  X = X_all,
  eta = 2,
  lp_y = lp_y_all,
  nlp_y  = nlp_y_all,
  align = FALSE,
  lp_w = lp_w_all,
  nlp_w = nlp_w_all,
  lambda = c(1000,2000,3000),
  delta = c(0.5,0.5),
  psi = 1,
  sigma_w = 1,
  sigma_y = 2,
  censor_rate = 0.1
)
```

intree *Interpreting Tree Ensembles with inTrees*

Description

The inTrees (interpretable trees) framework that extracts, measures, prunes and selects rules from a tree ensemble. All the codes we use are from the inTrees github repository to act as a work around method since package inTrees was removed from the CRAN repository.

Usage

```
intree(X, Y, ntree, typeDecay = 2, digits, n_rule)
```

Arguments

X	A matrix indicating the predictor variables.
Y	A response vector. If a factor, classification is assumed, otherwise regression is assumed.
ntree	Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
typeDecay	An integer of 1 or 2. 1 representing relative error and 2 representing error. The default is set to 2.
digits	An integer indicating the digits for rounding in Intrees.
n_rule	An integer indicating the minimum number of rules to consider in Intrees.

Value

A matrix including a set of relevant and non-redundant rules, and their metrics

Examples

```
X <- within(iris,rm("Species")); Y <- iris[,"Species"]
intree_result <- intree(X, Y, ntree=100, digits = 3, n_rule = 2000)
```

`plot.riAFTBART_estimate` *Plot the trace plots for the parameters from a fitted riAFT-BART model*

Description

This function creates the trace plots for the parameters from a fitted riAFT-BART model.

Usage

```
## S3 method for class 'riAFTBART_estimate'
plot(x, focus = "sigma", id = NULL, ...)
```

Arguments

`x`	A fitted object of from riAFTBART_fit function.
`focus`	A character specifying which parameter to plot.
`id`	A numeric vector indicating the subject or cluster index to plot, when the object to plot is random intercepts or predicted log survival time.
`...`	further arguments passed to or from other methods.

Value

A plot

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = stats::rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
```

```
Y = pmin(y,C)
status = as.numeric(y<=C)
res <- riAFTBART_fit(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
                     y.train = Y, trt.train = trt.train, trt.test = trt.test,
                     x.train = cbind(x1,x2),
                     x.test = cbind(x1,x2),
                     cluster.id = cluster.id)
plot(x = res, focus = "sigma")
```

---

plot.riAFTBART_survProb

Plot the fitted survival curves from riAFT-BART model

Description

This function plot the mean/individual survival curves from a fitted riAFT-BART model

Usage

```
## S3 method for class 'riAFTBART_survProb'
plot(x, test.only = FALSE, train.only = TRUE, id = NULL, ...)
```

Arguments

`x`	An object from cal_surv_prob() function.
`test.only`	A logical indicating whether or not only data from the test set should be computed. The default is FALSE.
`train.only`	A logical indicating whether or not only data from the training set should be computed. The default is FALSE.
`id`	A vector representing the IDs for the individual survival curves to plot. The default is NULL and the mean survival curves will be plotted.
`...`	further arguments passed to or from other methods.

Value

A plot

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
```

```
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = stats::rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
Y = pmin(y,C)
status = as.numeric(y<=C)
res <- riAFTBART_fit(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
                      y.train = Y, trt.train = trt.train, trt.test = trt.test,
                      x.train = cbind(x1,x2),
                      x.test = cbind(x1,x2),
                      cluster.id = cluster.id)
surv_prob_res <- cal_surv_prob(object = res, time.points = sort(exp(logtime)),
test.only = TRUE, cluster.id = cluster.id)
plot(x = surv_prob_res, test.only = TRUE, train.only = FALSE)
```

plot_gps *Plot the propensity score by treatment*

Description

This function estimates the propensity score for each treatment group and then plot the propensity score by each treatment to check covariate overlap.

Usage

```
plot_gps(trt, X, cluster.id, method = "Multinomial")
```

Arguments

trt	A numeric vector representing the treatment groups.
X	A dataframe or matrix, including all the covariates but not treatments, with rows corresponding to observations and columns to variables.
cluster.id	A vector of integers representing the clustering id. The cluster id should be an integer and start from 1.
method	A character indicating how to estimate the propensity score. The default is "Multinomial", which uses multinomial regression to estimate the propensity score.

Value

A plot

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
plot_gps(trt = trt.train, X = cbind(x1, x2), cluster.id = cluster.id)
```

`riAFTBART` *A flexible approach for causal inference with multiple treatments and clustered survival outcomes*

Description

This function implements the random effect accelerated failure time BART (riAFT-BART) for causal inference with multiple treatments and clustered survival outcomes.

Usage

```
riAFTBART(
  M.burnin,
  M.keep,
  M.thin = 1,
  status,
  y,
  x,
  trt,
  cluster.id,
  verbose = FALSE,
  estimand = "ATE",
  reference_trt = NULL
)
```

Arguments

`M.burnin`	A numeric value indicating the number of MCMC iterations to be treated as burn in.
`M.keep`	A numeric value indicating the number of MCMC posterior draws after burn in.
`M.thin`	A numeric value indicating the thinning parameter.
`status`	A vector of event indicators: status = 1 indicates that the event was observed while status = 0 indicates the observation was right-censored.
`y`	A vector of follow-up times.
`x`	A dataframe or matrix, including all the covariates but not treatments with rows corresponding to observations and columns to variables.
`trt`	A numeric vector representing the treatment groups.
`cluster.id`	A vector of integers representing the clustering id. The cluster id should be an integer and start from 1.
`verbose`	A logical indicating whether to show the progress bar for riAFT-BART. The default is FALSE
`estimand`	A character string representing the type of causal estimand. Only `"ATT"` or `"ATE"` is allowed. When the `estimand` = `"ATT"`, users also need to specify the reference treatment group by setting the `reference_trt` argument.
`reference_trt`	A numeric value indicating reference treatment group for ATT effect.

Value

A list of causal estimands in terms of log T between different treatment groups.

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = stats::rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
```

```
Y = pmin(y,C)
status = as.numeric(y<=C)
res_ate <- riAFTBART(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
                     y = Y, trt = trt.train,
                     x = cbind(x1,x2),
                     cluster.id = cluster.id, estimand = "ATE")
```

---

riAFTBART_fit   *Fit a random effect accelerated failure time BART model*

Description

This function implements the random effect accelerated failure time BART (riAFT-BART) algorithm.

Usage

```
riAFTBART_fit(
  M.burnin,
  M.keep,
  M.thin = 1,
  status,
  y.train,
  x.train,
  trt.train,
  x.test,
  trt.test,
  cluster.id,
  verbose = FALSE,
  SA = FALSE,
  prior_c_function_used = NULL,
  gps = NULL
)
```

Arguments

`M.burnin`	A numeric value indicating the number of MCMC iterations to be treated as burn in.
`M.keep`	A numeric value indicating the number of MCMC posterior draws after burn in.
`M.thin`	A numeric value indicating the thinning parameter.
`status`	A vector of event indicators: status = 1 indicates that the event was observed while status = 0 indicates the observation was right-censored.
`y.train`	A vector of follow-up times.
`x.train`	A dataframe or matrix, including all the covariates but not treatments for training data, with rows corresponding to observations and columns to variables.

`trt.train`	A numeric vector representing the treatment groups for the training data. If there's no treatment indicator, then set to `NULL`.
`x.test`	A dataframe or matrix, including all the covariates but not treatments for testing data, with rows corresponding to observations and columns to variables.
`trt.test`	A numeric vector representing the treatment groups for the testing data. If there's no treatment indicator, then set to `NULL`.
`cluster.id`	A vector of integers representing the clustering id. The cluster id should be an integer and start from 1.
`verbose`	A logical indicating whether to show the progress bar. The default is FALSE
`SA`	A logical indicating whether to conduct sensitivity analysis. The default is FALSE.
`prior_c_function_used`	Prior confounding functions used for SA, which is inherited from the sa function. The default is NULL.
`gps`	Generalized propensity score, which is inherited from the sa function. The default is NULL.

Value

A list with the following elements:

`b:`	A matrix including samples from the posterior of the random effects.
`tree:`	A matrix with M.keep rows and nrow(x.train) columns represnting the predicted log survival time for x.train.
`tree.pred:`	A matrix with M.keep rows and nrow(x.test) columns represnting the predicted log survival time for x.test.
`tau:`	A vector representing the posterior samples of tau, the standard deviation of the random effects.
`sigma:`	A vector representing the posterior samples of sigma, the residual/error standard deviation.
`vip:`	A matrix with M.keep rows and ncol(x.train) columns represnting the variable inclusion proportions for each variable.

Examples

```
library(riAFTBART)
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = stats::rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
sig.error = 0.5
```

```
censoring.rate = 0.02
x1 = stats::rnorm(N,0.5,1)
x2 = stats::rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = stats::rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
Y = pmin(y,C)
status = as.numeric(y<=C)
res <- riAFTBART_fit(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
                      y.train = Y, trt.train = trt.train, trt.test = trt.test,
                      x.train = cbind(x1,x2),
                      x.test = cbind(x1,x2),
                      cluster.id = cluster.id)
```

sa   *Flexible Monte Carlo sensitivity analysis for unmeasured confounding*

Description

This function implements the flexible sensitivity analysis approach for unmeasured confounding with multiple treatments from multilevel survival data.

Usage

```
sa(
  M.burnin,
  M.keep,
  M.thin = 1,
  status,
  y.train,
  x.train,
  trt.train,
  x.test,
  trt.test,
  cluster.id,
  verbose = FALSE,
  formula = NULL,
  prior_c_function,
  Q1,
  Q2 = NULL,
  nCores = 1,
  ...
)
```

Arguments

`M.burnin`	A numeric value indicating the number of MCMC iterations to be treated as burn in.
`M.keep`	A numeric value indicating the number of MCMC posterior draws after burn in.
`M.thin`	A numeric value indicating the thinning parameter.
`status`	A vector of event indicators: status = 1 indicates that the event was observed while status = 0 indicates the observation was right-censored.
`y.train`	A vector of follow-up times.
`x.train`	A dataframe or matrix, including all the covariates but not treatments for training data, with rows corresponding to observations and columns to variables.
`trt.train`	A numeric vector representing the treatment groups for the training data.
`x.test`	A dataframe, including all the covariates but not treatments for testing data, with rows corresponding to observations and columns to variables.
`trt.test`	A numeric vector representing the treatment groups for the testing data.
`cluster.id`	A vector of integers representing the clustering id.
`verbose`	A logical indicating whether to show the progress bar. The default is FALSE
`formula`	A `formula` object for the analysis. The default is to use all terms specified in `x.train`.
`prior_c_function`	1) A vector of characters indicating the prior distributions for the confounding functions. Each character contains the random number generation code from the standard probability `distributions` in the `stats` package. 2) A vector of characters including the grid specifications for the confounding functions. It should be used when users want to formulate the confounding functions as scalar values. 3) A matrix indicating the point mass prior for the confounding functions
`Q1`	A numeric value indicating the number of draws of the GPS from the posterior predictive distribution
`Q2`	A numeric value indicating the number of draws from the prior distributions of the confounding functions
`nCores`	A numeric value indicating number of cores to use for parallel computing.
`...`	Other parameters that can be passed to BART functions

Value

A list with the following elements:

`result_riAFTBART:`	Corrected log survival time for the test data from the riAFT-BART model.
`c_functions:`	The confounding functions sampled from the specified distribution used in the analysis.

Examples

```
set.seed(20181223)
n = 5       # number of clusters
k = 50      # cluster size
N = n*k     # total sample size
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
beta3 = -2
sig.error = 0.5
censoring.rate = 0.02
x1 = rnorm(N,0.5,1)
x2 = rnorm(N,1.5,0.5)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
error = rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate) # censoring times
Y = pmin(y,C)
status = as.numeric(y<=C)
res_sa <- sa(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
             y.train = Y,trt.train = trt.train,trt.test = trt.test,
             x.train = cbind(x1,x2),
             x.test = cbind(x1,x2),
             cluster.id = cluster.id, verbose = F,prior_c_function = c(
               "runif(-0.6, 0)",# c(1,2)
               "runif(0, 0.6)",# c(2,1)
               "runif(-0.6, 0)", # c(2,3)
               "seq(-0.6, 0, by = 0.3)", # c(1,3)
               "seq(0, 0.6, by = 0.3)", # c(3,1)
              "runif(0, 0.6)" # c(3,2)
            ),Q1 = 1, nCores = 1)
```

---

var_select *Perform Variable Selection using Three Threshold-based Procedures*

Description

Performs variable selection with ri-AFTBART using the three thresholding methods introduced in Bleich et al. (2013).

Usage

```
var_select(
  M.burnin,
  M.keep,
  M.thin = 1,
  status,
  y.train,
  x.train,
  trt.train,
  x.test,
  trt.test,
  cluster.id,
  verbose = FALSE,
  n_permuate,
  alpha = 0.1,
  seed = NULL
)
```

Arguments

`M.burnin`	A numeric value indicating the number of MCMC iterations to be treated as burn in.
`M.keep`	A numeric value indicating the number of MCMC posterior draws after burn in.
`M.thin`	A numeric value indicating the thinning parameter.
`status`	A vector of event indicators: status = 1 indicates that the event was observed while status = 0 indicates the observation was right-censored.
`y.train`	A vector of follow-up times.
`x.train`	A dataframe or matrix, including all the covariates but not treatments for training data, with rows corresponding to observations and columns to variables.
`trt.train`	A numeric vector representing the treatment groups for the training data.
`x.test`	A dataframe or matrix, including all the covariates but not treatments for testing data, with rows corresponding to observations and columns to variables.
`trt.test`	A numeric vector representing the treatment groups for the testing data.
`cluster.id`	A vector of integers representing the clustering id. The cluster id should be an integer and start from 1.
`verbose`	A logical indicating whether to show the progress bar. The default is FALSE.
`n_permuate`	Number of permutations of the event time together with the censoring indicator to generate the null permutation distribution.
`alpha`	Cut-off level for the thresholds.
`seed`	An optional integer value to set the random seed for reproducibility. Default is NULL.

Value

A list with the following elements:

`var_local_selected:`
A character vector including all the variables selected using Local procedure.

`var_max_selected`:	A character vector including all the variables selected using Global Max procedure.
`var_global_se_selected`:	A character vector including all the variables selected using Global SE procedure.
`vip_perm`:	The permutation distribution for the variable inclusion proportions generated by permuting the event time together with the censoring indicator.
`vip_obs`:	The variable inclusion proportions for the actual data.

Examples

```
set.seed(20181223)
n = 2
k = 50
N = n*k
cluster.id = rep(1:n, each=k)
tau.error = 0.8
b = rnorm(n, 0, tau.error)
alpha = 2
beta1 = 1
beta2 = -1
beta3 = -2
sig.error = 0.5
censoring.rate = 0.02
x1 = rnorm(N,0.5,1)
x2 = rnorm(N,1.5,0.5)
error = rnorm(N,0,sig.error)
logtime = alpha + beta1*x1 + beta2*x2 + b[cluster.id] + error
y = exp(logtime)
C = rexp(N, rate=censoring.rate)
Y = pmin(y,C)
status = as.numeric(y<=C)
trt.train = sample(c(1,2,3), N, prob = c(0.4,0.3,0.2), replace = TRUE)
trt.test = sample(c(1,2,3), N, prob = c(0.3,0.4,0.2), replace = TRUE)
res <- var_select(M.burnin = 10, M.keep = 10, M.thin = 1, status = status,
                      y.train = Y, trt.train = trt.train, trt.test = trt.test,
                      x.train = cbind(x1,x2),
                      x.test = cbind(x1,x2),
                      cluster.id = cluster.id,
                      n_permuate = 4,alpha = 0.1,seed = 20181223)
```

Index