Package: quantoptr (via r-universe)

August 22, 2024

Type Package

Title Algorithms for Quantile- And Mean-Optimal Treatment Regimes

Author Yu Zhou [cre, aut], Lan Wang [ctb], Ben Sherwood [ctb], Rui Song [ctb]

Maintainer Yu Zhou <zhou0269@umn.edu>

Description Estimation methods for optimal treatment regimes under three different criteria, namely marginal quantile, marginal mean, and mean absolute difference. For the first two criteria, both one-stage and two-stage estimation method are implemented. A doubly robust estimator for estimating the quantile-optimal treatment regime is also included.

Version 0.1.3

License GPL (>= 2)

LazyData TRUE

Imports stringr, rgenoud (>= 5.7), quantreg (>= 5.18), parallel, methods, Rdpack

Depends R (>= 3.2), stats, utils

RoxygenNote 6.0.1

NeedsCompilation no

RdMacros Rdpack

Date/Publication 2018-02-05 05:56:14 UTC

Repository https://yuzhou-isabella.r-universe.dev

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

RemoteRef HEAD

RemoteSha 5bdd2c40cdc113eb818d6fc4923ac87710545b53

Contents

`abso_diff_est` *Estimate the Gini's mean difference/mean absolute difference(MAD) for a Given Treatment Regime*

Description

Estimate the MAD if the entire population follows a treatment regime indexed by the given parameters. This function supports the `IPWE_MADopt` function.

Usage

```
abso_diff_est(beta, x, y, a, prob, Cnobs)
```

Arguments

`beta`	a vector indexing the treatment regime. It indexes a linear treatment regime: $$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\}.$$
`x`	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear. This is important that columns in `x` matches with `beta`.
`y`	a vector, the observed responses from a sample
`a`	a vector of 0s and 1s, the observed treatments from a sample
`prob`	a vector, the propensity scores of getting treatment 1 in the samples
`Cnobs`	A matrix with two columns, enumerating all possible combinations of pairs of indexes. This can be generated by `combn(1:n, 2)`, where `n` is the number of unique observations.

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association*.

See Also

The function IPWE_MADopt is based on this function.

Examples

```
library(stats)
GenerateData.MAD <- function(n)
{
  x1 <- runif(n)
  x2 <- runif(n)
  tp <- exp(-1+1*(x1+x2))/(1+exp(-1+1*(x1+x2)))
  a<-rbinom(n = n, size = 1, prob=tp)
  error <- rnorm(length(x1))
  y <- (1 + a*0.3*(-1+x1+x2<0) +  a*-0.3*(-1+x1+x2>0)) * error
  return(data.frame(x1=x1,x2=x2,a=a,y=y))
}



n <- 500
testData <- GenerateData.MAD(n)
logistic.model.tx <- glm(formula = a~x1+x2, data = testData, family=binomial)
ph <- as.vector(logistic.model.tx$fit)
Cnobs <- combn(1:n, 2)
abso_diff_est(beta=c(1,2,-1),
              x=model.matrix(a~x1+x2, testData),
              y=testData$y,
              a=testData$a,
              prob=ph,
              Cnobs = Cnobs)
```

---

augX *Generate Pseudo-Responses Based on Conditional Quantile Regression Models*

Description

This function supports the DR_Qopt function. For every observation, we generate pseudo-observations corresponding to treatment 0 and 1 respectively based on working conditional quantile models.

Usage

```
augX(raw.data, length.out = 200, txVec, moCondQuant_0, moCondQuant_1,
  nlCondQuant_0 = FALSE, nlCondQuant_1 = FALSE, start_0 = NULL,
  start_1 = NULL, clnodes)
```

Arguments

`raw.data`	A data frame, must contain all the variables that appear in `moCondQuant_0` and `moCondQuant_1`.
`length.out`	an integer greater than 1. If one of the conditional quantile model is set to be nonlinear, this argument will be triggered and we will fit `length.out` models across quantiles equally spaced between 0.001 and 0.999. The larger this value, the more refined the performance of this method. Default is 200.
`txVec`	a numeric vector of observed treatment levels coded 0L and 1L.
`moCondQuant_0`	A formula, used to specify the formula for the conditional quantile function when treatment = 0.
`moCondQuant_1`	A formula, used to specify the formula for the conditional quantile function when treatment = 1.
`nlCondQuant_0`	logical. When `nlCondQuant_0` = `TRUE`, it is indicated that `moCondQuant_0` is nonlinear. The default value of this variable is `FALSE`.
`nlCondQuant_1`	logical. When `nlCondQuant_1` = `TRUE`, it is indicated that `moCondQuant_1` is nonlinear. The default value of this variable is `FALSE`.
`start_0`	either a list object, providing the starting value in estimating the parameters in the nonlinear conditional quantile model, given that treatment=0. Default is `NULL`, corresponding to the case when `nlCondQuant_0=FALSE`.
`start_1`	either a list object, providing the starting value in estimating the parameters in the nonlinear conditional quantile model, given that treatment=0. Default is `NULL`, corresponding to the case when `nlCondQuant_1=FALSE`.
`clnodes`	Either a cluster object to enable parallel computation or `NULL`. If `NULL`, no parallel computation will be used.

Details

This function implements the algorithm to generate individual level pseudo responses for two treatment levels respectively.

For each observation, two independent random variables from $unif[0,1]$ are generated. Denote them by u_0 and u_1. Approximately, this function then estimates the u_0th quantile of this observation were treatment level 0 is applied via the conditional u_0th quantile regression. This estimated quantile will be the pseudo-response for treatment 0. Similarly, this function the pseudo-response for treatment 1 will be estimated and returned.

See the reference paper for a more formal explanation.

Value

It returns a list object, consisting of the following elements:

1. `y.a.0`, the vector of estimated individual level pseudo outcomes, given the treatment is 0;
2. `y.a.1`, the vector of estimated individual level pseudo outcomes, given the treatment is 1;
3. `nlCondQuant_0`, logical, indicating whether the `y.a.0` is generated based on a nonlinear conditional quantile model.
4. `nlCondQuant_1`, logical, indicating whether the `y.a.1` is generated based on a nonlinear conditional quantile model.

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association*.

Examples

```
ilogit <- function(x) exp(x)/(1 + exp(x))
GenerateData.DR <- function(n)
{
  x1 <- runif(n,min=-1.5,max=1.5)
  x2 <- runif(n,min=-1.5,max=1.5)
  tp <- ilogit( 1 - 1*x1^2 - 1* x2^2)
  a <-rbinom(n,1,tp)
  y <- a * exp(0.11 - x1- x2) + x1^2 + x2^2 +  a*rgamma(n, shape=2*x1+3, scale = 1) +
       (1-a)*rnorm(n, mean = 2*x1 + 3, sd = 0.5)
  return(data.frame(x1=x1,x2=x2,a=a,y=y))
}
regimeClass = as.formula(a ~ x1+x2)
moCondQuant_0 = as.formula(y ~ x1+x2+I(x1^2)+I(x2^2))
moCondQuant_1 = as.formula(y ~ exp( 0.11 - x1 - x2)+ x1^2 + p0 + p1*x1
+ p2*x1^2 + p3*x1^3 +p4*x1^4 )
start_1 = list(p0=0, p1=1.5, p2=1, p3 =0,p4=0)

## Not run:
n<-200
testdata <- GenerateData.DR(n)
fit1 <- augX(raw.data=testdata, txVec = testdata$a,
             moCondQuant_0=moCondQuant_0, moCondQuant_1=moCondQuant_1,
             nlCondQuant_0=FALSE,   nlCondQuant_1=TRUE,
             start_1=start_1,
             clnodes=NULL)

# How to use parallel computing in AugX(): ##

# on Mac OSX/linux
 clnodes <- parallel::makeForkCluster(nnodes =getOption("mc.cores",2))
 fit2 <- augX(raw.data=testdata, length.out = 5, txVec = testdata$a,
             moCondQuant_0=moCondQuant_0, moCondQuant_1=moCondQuant_1,
             nlCondQuant_0=FALSE,   nlCondQuant_1=TRUE,
             start_1=start_1,
             clnodes=clnodes)

# on Windows
 clnodes <- parallel::makeCluster(2, type="PSOCK")
 fit3 <- augX(raw.data=testdata, length.out = 5, txVec = testdata$a,
             moCondQuant_0=moCondQuant_0, moCondQuant_1=moCondQuant_1,
             nlCondQuant_0=FALSE,   nlCondQuant_1=TRUE,
             start_1=start_1,
             clnodes=clnodes)

## End(Not run)
```

DR_Qopt	*The Doubly Robust Estimator of the Quantile-Optimal Treatment Regime*

Description

DR_Qopt implements the doubly robust estimation method to estimate the quantile-optimal treatment regime. The double robustness property means that it is consistent when either the propensity score model is correctly specified, or the conditional quantile function is correctly specified. Both linear and nonlinear conditional quantile models are considered. See 'Examples' for an illustrative example.

Usage

```
DR_Qopt(data, regimeClass, tau, moPropen = "BinaryRandom",
  nlCondQuant_0 = FALSE, nlCondQuant_1 = FALSE, moCondQuant_0,
  moCondQuant_1, max = TRUE, length.out = 200, s.tol, it.num = 8,
  cl.setup = 1, p_level = 1, pop.size = 3000, hard_limit = FALSE,
  start_0 = NULL, start_1 = NULL)
```

Arguments

data	a data frame, must contain all the variables that appear in moPropen, RegimeClass, moCondQuant_0, moCondQuant_1, and a column named y as the observed response.
regimeClass	a formula specifying the class of treatment regimes to search, e.g. if regimeClass = a~x1+x2, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported. See also 'Details'.
tau	a value between 0 and 1. This is the quantile of interest.
moPropen	The propensity score model for the probability of receiving treatment level 1. When moPropen equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. a1~x1.
nlCondQuant_0	Logical. When nlCondQuant_0=TRUE, this means the prespecified model for the conditional quantile function given a=0 is nonlinear, so the provided moCondQuant_0 should be nonlinear.
nlCondQuant_1	Logical. When nlCondQuant_1=TRUE, this means the prespecified model for the conditional quantile function given a=1 is nonlinear, so the provided moCondQuant_1 should be nonlinear.

`moCondQuant_0`	Either a formula or a string representing the parametric form of the conditional quantile function given that treatment=0.
`moCondQuant_1`	Either a formula or a string representing the parametric form of the conditional quantile function given that treatment=1.
`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal quantile; if `max=FALSE`, we wish to minimize the marginal quantile. The default is `TRUE`.
`length.out`	an integer greater than 1. If one of the conditional quantile model is set to be nonlinear, this argument will be triggered and we will fit `length.out` models across quantiles equally spaced between 0.001 and 0.999. Default is 200.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when `genoud` stops. Default is `FALSE`.
`start_0`	a named list or named numeric vector of starting estimates for the conditional quantile function when `treatment = 0`. This is required when `nlCondQuant_0=TRUE`.
`start_1`	a named list or named numeric vector of starting estimates for the conditional quantile function when `treatment = 1`. This is required when `nlCondQuant_1=TRUE`.

Details

- Standardization on covariates AND explanation on the differences between the two returned regime parameters.

  Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

  This estimated parameters indexing the quantile-optimal treatment regime are returned *in two scales:*

  1. The returned `coefficients` is the set of parameters after covariates X are standardized to be in the interval [0, 1]. To be exact, every covariate is subtracted by the smallest observed value and divided by the difference between the largest and the smallest value. Next, we carried out the algorithm in Wang 2016 to get the estimated regime parameters, `coefficients`, based on the standardized data. For the identifiability issue, we force the Euclidean norm of `coefficients` to be 1.

2. In contrast, `coef.orgn.scale` corresponds to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

$$d(x) = I\{\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_k x_k > 0\}.$$

The estimated $\hat{\boldsymbol{\beta}}$ is returned as `coef.orgn.scale`. The same as `coefficients`, we force the Euclidean norm of `coef.orgn.scale` to be 1.

If, for each input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coefficients` and `coef.orgn.scale` will render identical.

- Property of the doubly robust(DR) estimator. The DR estimator `DR_Qopt` is consistent if either the propensity score model or the conditional quantile regression model is correctly specified. (Wang et. al. 2016)

Value

This function returns an object with 9 objects. Both `coefficients` and `coef.orgn.scale` were normalized to have unit euclidean norm.

`coefficients` the parameters indexing the estimated quantile-optimal treatment regime for standardized covariates.

`coef.orgn.scale` the parameter indexing the estimated quantile-optimal treatment regime for the original input covariates.

`tau` the quantile of interest

`hatQ` the estimated marginal tau-th quantile when the treatment regime indexed by `coef.orgn.scale` is applied on everyone. See the 'details' for connection between `coef.orgn.scale` and `coefficient`.

`call` the user's call.

`moPropen` the user specified propensity score model

`regimeClass` the user specified class of treatment regimes

`moCondQuant_0` the user specified conditional quantile model for treatment 0

`moCondQuant_1` the user specified conditional quantile model for treatment 1

Author(s)

Yu Zhou, `<zhou0269@umn.edu>`

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association.*

See Also

`dr_quant_est`, `augX`

Examples

```
ilogit <- function(x) exp(x)/(1 + exp(x))
GenerateData.DR <- function(n)
{
 x1 <- runif(n,min=-1.5,max=1.5)
 x2 <- runif(n,min=-1.5,max=1.5)
 tp <- ilogit( 1 - 1*x1^2 - 1* x2^2)
 a <-rbinom(n,1,tp)
 y <- a * exp(0.11 - x1- x2) + x1^2 + x2^2 +  a*rgamma(n, shape=2*x1+3, scale = 1) +
 (1-a)*rnorm(n, mean = 2*x1 + 3, sd = 0.5)
 return(data.frame(x1=x1,x2=x2,a=a,y=y))
}

regimeClass <- as.formula(a ~ x1+x2)
moCondQuant_0 <- as.formula(y ~ x1+x2+I(x1^2)+I(x2^2))
moCondQuant_1 <- as.formula(y ~ exp( 0.11 - x1 - x2)+ x1^2 + p0 + p1*x1
                           + p2*x1^2 + p3*x1^3 +p4*x1^4 )
start_1 = list(p0=0, p1=1.5, p2=1, p3 =0,p4=0)



n <- 400
testdata <- GenerateData.DR(n)

## Examples below correctly specified both the propensity model and
##  the conditional quantile model.

 system.time(
 fit1 <- DR_Qopt(data=testdata, regimeClass = regimeClass,
                 tau = 0.25,
                 moPropen = a~I(x1^2)+I(x2^2),
                 moCondQuant_0 = moCondQuant_0,
                 moCondQuant_1 = moCondQuant_1,
                 nlCondQuant_1 = TRUE,  start_1=start_1,
                 pop.size = 1000))
 fit1
 ## Go parallel for the same fit. It would save a lot of time.
 ### Could even change the cl.setup to larger values
 ### if more cores are available.

 system.time(fit2 <- DR_Qopt(data=testdata, regimeClass = regimeClass,
                 tau = 0.25,
                 moPropen = a~I(x1^2)+I(x2^2),
                 moCondQuant_0 = moCondQuant_0,
                 moCondQuant_1 = moCondQuant_1,
                 nlCondQuant_1 = TRUE,  start_1=start_1,
                 pop.size = 1000, cl.setup=2))
 fit2
```

dr_quant_est — *The Doubly Robust Quantile Estimator for a Given Treatment Regime*

Description

Given a fixed treatment regime, this doubly robust estimator estimates the marginal quantile of responses when it is followed by every unit in the target population. It took advantages of conditional quantile functions for different treatment levels when they are available.

Usage

```
dr_quant_est(beta, x, y, a, prob, tau, y.a.0, y.a.1, num_min = FALSE)
```

Arguments

`beta`	a vector indexing the treatment regime. It indexes a linear treatment regime: $$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\}.$$
`x`	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear. This is important that columns in `x` matches with `beta`.
`y`	a vector, the observed responses from a sample
`a`	a vector of 0s and 1s, the observed treatments from a sample
`prob`	a vector, the propensity scores of getting treatment 1 in the samples
`tau`	The quantile of interest
`y.a.0`	Estimated conditional potential outcome given that treatment = 0, which can be calculated by the function `augX`.
`y.a.1`	Estimated conditional potential outcome given that treatment = 1, which can be calculated by the function `augX`.
`num_min`	logical. If `TRUE`, the number of global minimizers for the objective function is returned.

Details

The double robustness property means that it can consistently estimate the marginal quantile when either the propensity score model is correctly specified, or the conditional quantile function is correctly specified.

See Also

`augX`

`get_os` *Get the OS from R*

Description

Get the type of the operating system. The returned value is used in configuring parallel computation for the implemented algorithms.

Usage

```
get_os()
```

References

This function is adapted from https://www.r-bloggers.com/identifying-the-os-from-r/

`IPWE_MADopt` *Estimation of the Optimal Treatment Regime defined as Minimizing Gini's Mean Differences*

Description

`IPWE_MADopt` seeks to estimated the treatment regime which **minimizes** the Gini's Mean difference defined below.

Besides mean and quantile criterion, in some applications people seek minimization of dispersion in the outcome, which, for example, can be described by Gini's mean difference. Formally, it is defined as the absolute differences of two random variables Y_1 and Y_2 drawn independently from the same distribution:

$$MAD := E(|Y_1 - Y_2|).$$

Given a treatment regime d, define the potential outcome of a subject following the treatment recommended by `d` as $Y^*(d)$. When d is followed by everyone in the target population, the Gini's mean absolute difference is

$$MAD(d) := E(|Y_1^*(d) - Y_2^*(d)|).$$

Usage

```
IPWE_MADopt(data, regimeClass, moPropen = "BinaryRandom", s.tol, it.num = 8,
  hard_limit = FALSE, cl.setup = 1, p_level = 1, pop.size = 3000)
```

Arguments

`data`	a data frame, containing variables in the `moPropen` and `RegimeClass` and a component `y` as the response.
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported. See also 'Details'.
`moPropen`	The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when `genoud` stops. Default is `FALSE`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).

Details

Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

This estimated parameters indexing the MAD-optimal treatment regime are returned *in two scales:*

1. The returned `coefficients` is the set of parameters after covariates X are standardized to be in the interval [0, 1]. To be exact, every covariate is subtracted by the smallest observed value and divided by the difference between the largest and the smallest value. Next, we carried out the algorithm in Wang et al. 2017 to get the estimated regime parameters, `coefficients`, based on the standardized data. For the identifiability issue, we force the Euclidean norm of `coefficients` to be 1.

2. In contrast, `coef.orgn.scale` corresponds to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

$$d(x; \hat{\boldsymbol{\beta}}) = I\{\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_k x_k > 0\}.$$

   The estimated $\hat{\boldsymbol{\beta}}$ is returned as `coef.orgn.scale`. The same as `coefficients`, we force the Euclidean norm of `coef.orgn.scale` to be 1.

If, for every input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coefficients` and `coef.orgn.scale` will render identical.

Value

This function returns an object with 6 objects. Both `coefficients` and `coef.orgn.scale` were normalized to have unit euclidean norm.

`coefficients` the parameters indexing the estimated MAD-optimal treatment regime for standardized covariates.

`coef.orgn.scale` the parameter indexing the estimated MAD-optimal treatment regime for the original input covariates.

`hat_MAD` the estimated MAD when a treatment regime indexed by `coef.orgn.scale` is applied on everyone. See the 'details' for connection between `coef.orgn.scale` and `coefficient`.

`call` the user's call.

`moPropen` the user specified propensity score model

`regimeClass` the user specified class of treatment regimes

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association.*

Examples

```
GenerateData.MAD <- function(n)
{
  x1 <- runif(n)
  x2 <- runif(n)
  tp <- exp(-1+1*(x1+x2))/(1+exp(-1+1*(x1+x2)))
  a<-rbinom(n = n, size = 1, prob=tp)
  error <- rnorm(length(x1))
  y <- (1 + a*0.6*(-1+x1+x2<0) +  a*-0.6*(-1+x1+x2>0)) * error
  return(data.frame(x1=x1,x2=x2,a=a,y=y))
}
# The true MAD optimal treatment regime for this generative model
# can be deduced trivially, and it is:  c( -0.5773503,  0.5773503,  0.5773503).


# With correctly specified propensity model   ####
```

```
n <- 400
testData <- GenerateData.MAD(n)
fit1 <- IPWE_MADopt(data = testData, regimeClass = a~x1+x2,
                    moPropen=a~x1+x2, cl.setup=2)
fit1




# With incorrectly specified propensity model ####

fit2 <- IPWE_MADopt(data = testData, regimeClass = a~x1+x2,
                    moPropen="BinaryRandom", cl.setup=2)
fit2
```

`IPWE_Mopt` *Estimate the Mean-optimal Treatment Regime*

Description

`IPWE_Mopt` aims at estimating the treatment regime which maximizes the marginal mean of the potential outcomes.

Usage

```
IPWE_Mopt(data, regimeClass, moPropen = "BinaryRandom", max = TRUE,
  s.tol = 1e-04, cl.setup = 1, p_level = 1, it.num = 10,
  hard_limit = FALSE, pop.size = 3000)
```

Arguments

`data`	a data frame, containing variables in the `moPropen` and `RegimeClass` and a component `y` as the response.
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported. See also 'Details'.
`moPropen`	The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.

`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal mean; If `max=FALSE`, we wish to minimize the marginal mean. The default is `TRUE`.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when `genoud` stops. Default is `FALSE`.
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).

Details

Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

This functions returns the estimated parameters indexing the mean-optimal treatment regime under two scales.

The returned `coefficients` is the set of parameters when covariates are all standardized to be in the interval [0, 1] by subtracting the smallest observed value and divided by the difference between the largest and the smallest value.

While the returned `coef.orgn.scale` corresponds to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

$$d(x) = I\{\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_k x_k > 0\}.$$

The estimated $\hat{\boldsymbol{\beta}}$ is returned as `coef.orgn.scale`.

If, for every input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coefficients` and `coef.orgn.scale` will render identical.

Value

This function returns an object with 6 objects. Both `coefficients` and `coef.orgn.scale` were normalized to have unit euclidean norm.

`coefficients` the parameters indexing the estimated mean-optimal treatment regime for standardized covariates.

`coef.orgn.scale` the parameter indexing the estimated mean-optimal treatment regime for the original input covariates.

`hatM` the estimated marginal mean when a treatment regime indexed by `coef.orgn.scale` is applied on everyone. See the 'details' for connection between `coef.orgn.scale` and `coefficient`.

`call` the user's call.

`moPropen` the user specified propensity score model

`regimeClass` the user specified class of treatment regimes

Author(s)

Yu Zhou, <`zhou0269@umn.edu`>, with substantial contribution from Ben Sherwood.

References

Zhang B, Tsiatis AA, Laber EB and Davidian M (2012). "A robust method for estimating optimal treatment regimes." *Biometrics*, **68**(4), pp. 1010–1018.

Examples

```
GenerateData.test.IPWE_Mopt <- function(n)
{
  x1 <- runif(n)
  x2 <- runif(n)
  tp <- exp(-1+1*(x1+x2))/(1+exp(-1+1*(x1+x2)))
  error <- rnorm(length(x1), sd=0.5)
  a <- rbinom(n = n, size = 1, prob=tp)
  y <- 1+x1+x2 +  a*(3 - 2.5*x1 - 2.5*x2) +
        (0.5 + a*(1+x1+x2)) * error
  return(data.frame(x1=x1,x2=x2,a=a,y=y))
}

n <- 500
testData <- GenerateData.test.IPWE_Mopt(n)
fit <- IPWE_Mopt(data=testData, regimeClass = a~x1+x2,
                 moPropen=a~x1+x2,
                 pop.size=1000)
fit
```

`IPWE_Qopt` *Estimate the Quantile-optimal Treatment Regime*

Description

Estimate the Quantile-optimal Treatment Regime by inverse probability of weighting

Usage

```
IPWE_Qopt(data, regimeClass, tau, moPropen = "BinaryRandom", max = TRUE,
  s.tol, it.num = 8, hard_limit = FALSE, cl.setup = 1, p_level = 1,
  pop.size = 3000)
```

Arguments

`data`	a data frame, containing variables in the `moPropen` and `RegimeClass` and a component `y` as the response.
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported. See also 'Details'.
`tau`	a value between 0 and 1. This is the quantile of interest.
`moPropen`	The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal quantile; if `max=FALSE`, we wish to minimize the marginal quantile. The default is `TRUE`.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when `genoud` stops. Default is `FALSE`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).

Details

Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

This estimated parameters indexing the quantile-optimal treatment regime are returned *in two scales:*

1. The returned `coefficients` is the set of parameters after covariates X are standardized to be in the interval [0, 1]. To be exact, every covariate is subtracted by the smallest observed value and divided by the difference between the largest and the smallest value. Next, we carried out the algorithm in Wang et al. 2017 to get the estimated regime parameters, `coefficients`, based on the standardized data. For the identifiability issue, we force the Euclidean norm of `coefficients` to be 1.
2. In contrast, `coef.orgn.scale` corresponds to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

$$d(x) = I\{\hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_k x_k > 0\}.$$

The estimated $\hat{\boldsymbol{\beta}}$ is returned as `coef.orgn.scale`. The same as `coefficients`, we force the Euclidean norm of `coef.orgn.scale` to be 1.

If, for every input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coefficients` and `coef.orgn.scale` will render identical.

Value

This function returns an object with 7 objects. Both `coefficients` and `coef.orgn.scale` were normalized to have unit euclidean norm.

`coefficients` the parameters indexing the estimated quantile-optimal treatment regime for standardized covariates.

`coef.orgn.scale` the parameter indexing the estimated quantile-optimal treatment regime for the original input covariates.

`tau` the quantile of interest

`hatQ` the estimated marginal tau-th quantile when the treatment regime indexed by `coef.orgn.scale` is applied on everyone. See the 'details' for connection between `coef.orgn.scale` and `coefficient`.

`call` the user's call.

`moPropen` the user specified propensity score model

`regimeClass` the user specified class of treatment regimes

Author(s)

Yu Zhou, <`zhou0269@umn.edu`> with substantial contribution from Ben Sherwood.

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association*.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 0.5)
  tp <- exp(-1+1*(x1+x2))/(1+exp(-1+1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=tp)
  y <-  1+x1+x2 +  a*(3 - 2.5*x1 - 2.5*x2) +  (0.5 + a*(1+x1+x2)) * error
  return(data.frame(x1=x1,x2=x2,a=a,y=y))
}
n <- 300
testData <- GenerateData(n)

# 1. Estimate the 0.25th-quantile optimal treatment regime. ###

fit1 <- IPWE_Qopt(data = testData, regimeClass = "a~x1+x2",
           tau = 0.25, moPropen="a~x1+x2")
fit1


# 2. Go parallel. This saves time in calculation. ###

fit2 <- IPWE_Qopt(data = testData, regimeClass = "a~x1+x2",
           tau = 0.25, moPropen="a~x1+x2", cl.setup=2)
fit2




# 3. Set a quardratic term in the class #######################

fit3 <- IPWE_Qopt(data = testData, regimeClass = "a~x1+x2+I(x1^2)",
                  tau = 0.25, moPropen="a~x1+x2", pop.size=1000)
fit3


# 4. Set screen prints level. #######################
# Set the p_level to be 0,
# then all screen prints from the genetic algorithm will be suppressed.

fit4 <- IPWE_Qopt(data = testData, regimeClass = "a~x1+x2",
           tau = 0.25, moPropen="a~x1+x2", cl.setup=2, p_level=0)
fit4
```

---

mean_est *The Inverse Probability Weighted Estimator of the Marginal Mean Given a Specific Treatment Regime*

Description

Estimate the marginal mean of the response when the entire population follows a treatment regime. This function implements the inverse probability weighted estimator proposed by Baqun Zhang et. al..

This function supports the `mestimate` function.

Usage

```
mean_est(beta, x, a, y, prob)
```

Arguments

`beta`	a vector indexing the treatment regime. It indexes a linear treatment regime: $$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\}.$$
`x`	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear. This is important that columns in `x` matches with `beta`.
`a`	a vector of 0s and 1s, the observed treatments from a sample
`y`	a vector, the observed responses from a sample
`prob`	a vector, the propensity scores of getting treatment 1 in the samples

References

Zhang B, Tsiatis AA, Laber EB and Davidian M (2012). “A robust method for estimating optimal treatment regimes.” *Biometrics*, **68**(4), pp. 1010–1018.

`mestimate` *The Mean-Optimal Treatment Regime Wrapper Function*

Description

The wrapper function for mean-optimal treatment regime that calls a genetic algorithm. This function supports the `IPWE_Mopt` function.

Usage

```
mestimate(x, y, a, prob, p_level, nvars, hard_limit = FALSE, max = TRUE,
  cl.setup = 1, s.tol = 1e-04, it.num = 8, pop.size = 3000)
```

Arguments

x	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear.
y	a vector, the observed responses from a sample
a	a vector of 0s and 1s, the observed treatments from a sample
prob	a vector, the propensity scores of getting treatment 1 in the samples
p_level	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
nvars	an integer. The number of parameters indexing a treatment regime.
hard_limit	logical. This logical variable determines if the max.generations variable is a binding constraint for genoud.
max	logical. If max=TRUE, it indicates we wish to maximize the marginal mean; If max=FALSE, we wish to minimize the marginal mean. The default is TRUE.
cl.setup	the number of nodes. >1 indicates choosing parallel computing option in rgenoud::genoud. Default is 1.
s.tol	This is the tolerance level used by genoud. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating it.num.
it.num	integer > 1. This argument will be used in rgeound::geound function. If there is no improvement in the objective function in this number of generations, rgenoud::genoud will think that it has found the optimum.
pop.size	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (rgenoud::genoud).

References

Zhang B, Tsiatis AA, Laber EB and Davidian M (2012). “A robust method for estimating optimal treatment regimes.” *Biometrics*, **68**(4), pp. 1010–1018.

See Also

The function IPWE_Mopt is based on this function.

---

qestimate *The Quantile-Optimal Treatment Regime Wrapper Function*

Description

The wrapper function for quantile-optimal treatment regime that calls a genetic algorithm. This function supports the IPWE_Qopt function.

Usage

```
qestimate(tau, x, y, a, prob, p_level, nvars, hard_limit, max = TRUE,
  cl.setup = 1, s.tol = 1e-04, it.num = 8, pop.size = 3000)
```

Arguments

`tau`	a numeric value between 0 and 1. The quantile level of interest.
`x`	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear.
`y`	a vector, the observed responses from a sample
`a`	a vector of 0s and 1s, the observed treatments from a sample
`prob`	a vector, the propensity scores of getting treatment 1 in the samples
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`nvars`	an integer. The number of parameters indexing a treatment regime.
`hard_limit`	logical. This logical variable determines if the max.generations variable is a binding constraint for `rgenoud::genoud()`.
`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal quantile; if `max=FALSE`, we wish to minimize the marginal quantile. The default is `TRUE`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association*.

See Also

The function `IPWE_Qopt` is based on this function.

quant_est *Estimate the Marginal Quantile Given a Specific Treatment Regime*

Description

Estimate the marginal quantile if the entire population follows a treatment regime indexed by the given parameters. This function supports the qestimate function.

Usage

```
quant_est(beta, x, y, a, prob, tau)
```

Arguments

beta	a vector indexing the treatment regime. It indexes a linear treatment regime: $$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\}.$$
x	a matrix of observed covariates from the sample. Notice that we assumed the class of treatment regimes is linear. This is important that columns in x matches with beta.
y	a vector, the observed responses from a sample
a	a vector of 0s and 1s, the observed treatments from a sample
prob	a vector, the propensity scores of getting treatment 1 in the samples
tau	a numeric value between 0 and 1. The quantile level of interest.

TwoStg_Mopt *Estimate the Two-stage Mean-Optimal Treatment Regime*

Description

This function implements the estimator of two-stage mean-optimal treatment regime by inverse probability of weighting proposed by Baqun Zhang. As there are more than one stage, the second stage treatment regime could take into account the evolving status of an individual after the first stage and the treatment level received in the first stage. We assume the options at the two stages are both binary and take the form:

$$d_1(x_{stage1}) = I\left(\beta_{10} + \beta_{11}x_{11} + ... + \beta_{1k}x_{1k} > 0\right),$$

$$d_2(x_{stage2}) = I\left(\beta_{20} + \beta_{21}x_{21} + ... + \beta_{2j}x_{2j} > 0\right)$$

Usage

```
TwoStg_Mopt(data, regimeClass.stg1, regimeClass.stg2,
  moPropen1 = "BinaryRandom", moPropen2 = "BinaryRandom", max = TRUE,
  s.tol, cl.setup = 1, p_level = 1, it.num = 10, pop.size = 3000,
  hard_limit = FALSE)
```

Arguments

`data`	a data frame, containing variables in the `moPropen` and `RegimeClass` and a component `y` as the response.
`regimeClass.stg1`	a formula or a string specifying the Class of treatment regimes at stage 1, e.g. `a1~x1+x2`
`regimeClass.stg2`	a formula or a string specifying the Class of treatment regimes at stage 2, e.g. `a2~x1+a1+x2`
`moPropen1`	The propensity score model for the probability of receiving treatment level 1 at the first stage . When `moPropen1` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample at the first stage will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`moPropen2`	The propensity score model for the probability of receiving treatment level 1 at the second stage . When `moPropen2` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample at the second stage will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a2~x1+a1+x2`.
`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal mean; if `max=FALSE`, we wish to minimize the marginal mean. The default is `TRUE`.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when `genoud` stops. Default is `FALSE`.

Details

Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

For every stage `k`, $k = 1, 2$, this estimated parameters indexing the two-stage mean-optimal treatment regime are returned *in two scales:*

1. , the returned `coef.k` is the set of parameters that we estimated after standardizing every covariate available for decision-making at stage `k` to be in the interval [0, 1]. To be exact, every covariate is subtracted by the smallest observed value and divided by the difference between the largest and the smallest value. Next, we carried out the algorithm in Wang 2016 to get the estimated regime parameters, `coef.k`, based on the standardized data. For the identifiability issue, we force the Euclidean norm of `coef.k` to be 1.
2. The difference between `coef.k` and `coef.orgn.scale.k` is that the latter set of parameters correspond to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

   $$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\},$$

   where the β values are returned as `coef.orgn.scale.k`, and the the vector $(1, x_1, ..., x_k)$ corresponds to the specified class of treatment regimes in the `k`th stage.

If, for every input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coef.k` and `coef.orgn.scale.k` will render identical.

Value

This function returns an object with 6 objects. Both `coef.1`, `coef.2` and `coef.orgn.scale.1`, `coef.orgn.scale.2` were normalized to have unit euclidean norm.

`coef.1`, `coef.2` the set of parameters indexing the estimated mean-optimal treatment regime for standardized covariates.

`coef.orgn.scale.1`, `coef.orgn.scale.2` the set of parameter indexing the estimated mean-optimal treatment regime for the original input covariates.

`hatM` the estimated marginal mean when the treatment regime indexed by `coef.orgn.scale.1` and `coef.orgn.scale.2` is applied on the entire population. See the 'details' for connection between `coef.orgn.scale.k` and `coef.k`.

`call` the user's call.

`moPropen1`, `moPropen2` the user specified propensity score models for the first and the second stage respectively

`regimeClass.stg1`, `regimeClass.stg2` the user specified class of treatment regimes for the first and the second stage respectively

Author(s)

Yu Zhou, `<zhou0269@umn.edu>`

References

Zhang B, Tsiatis AA, Laber EB and Davidian M (2013). “Robust estimation of optimal dynamic treatment regimes for sequential treatment decisions.” *Biometrika*, **100**(3).

Examples

```
ilogit <- function(x) exp(x)/(1 + exp(x))
GenerateData.2stg <- function(n){
 x1 <- runif(n)
 p1 <- ilogit(-0.5+x1)
 a1 <- rbinom(n, size=1, prob=p1)

 x2 <- runif(n, x1, x1+1)
 p2 <- ilogit(-1 + x2)
 a2 <- rbinom(n, size=1, prob=p2)

 mean <- 1+x1+a1*(1-3*(x1-0.2)^2) +x2 + a2*(1-x2-x1)
 y <- mean + (1+a1*(x1-0.5)+0.5*a2*(x2-1))*rnorm(n,0,sd = 1)
 return(data.frame(x1,a1,x2,a2,y))
}

n <- 400
testdata <- GenerateData.2stg(n)

fit <- TwoStg_Mopt(data=testdata,
                   regimeClass.stg1="a1~x1", regimeClass.stg2="a2~x1+a1+x2",
                   moPropen1="a1~x1", moPropen2="a2~x2",
                   cl.setup=2)
fit

fit2 <- TwoStg_Mopt(data=testdata,
                   regimeClass.stg1="a1~x1", regimeClass.stg2="a2~a1+x1*x2",
                   moPropen1="a1~x1", moPropen2="a2~x2",
                   cl.setup=2)
fit2
```

`TwoStg_Qopt` *Estimate the Two-stage Quantile-optimal Treatment Regime*

Description

This function implements the estimator of two-stage quantile-optimal treatment regime by inverse probability of weighting proposed by Lan Wang, et al. As there are more than one stage, the second stage treatment regime could take into account the evolving status of an individual after the first stage and the treatment level received in the first stage. We assume the options at the two stages are both binary and take the form:

$$d_1(x) = I(\beta_{10} + \beta_{11}x_{11} + ... + \beta_{1k}x_{1k} > 0),$$

$$d_2(x) = I(\beta_{20} + \beta_{21}x_{21} + ... + \beta_{2p}x_{2p} > 0)$$

Usage

```
TwoStg_Qopt(data, tau, regimeClass.stg1, regimeClass.stg2,
  moPropen1 = "BinaryRandom", moPropen2 = "BinaryRandom", s.tol = 1e-04,
  it.num = 8, max = TRUE, cl.setup = 1, p_level = 1, pop.size = 1000,
  hard_limit = FALSE)
```

Arguments

`data`	a data frame, containing variables in the `moPropen` and `RegimeClass` and a component `y` as the response.
`tau`	a value between 0 and 1. This is the quantile of interest.
`regimeClass.stg1`	a formula or a string specifying the Class of treatment regimes at stage 1, e.g. `a1~x1+x2`
`regimeClass.stg2`	a formula or a string specifying the Class of treatment regimes at stage 2, e.g. `a2~x1+a1+x2`
`moPropen1`	The propensity score model for the probability of receiving treatment level 1 at the first stage . When `moPropen1` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample at the first stage will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`moPropen2`	The propensity score model for the probability of receiving treatment level 1 at the second stage . When `moPropen2` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample at the second stage will be employed as a good estimate of the probability for each observation. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a2~x1+a1+x2`.
`s.tol`	This is the tolerance level used by `genoud`. Default is 10^{-5} times the difference between the largest and the smallest value in the observed responses. This is particularly important when it comes to evaluating `it.num`.
`it.num`	integer > 1. This argument will be used in `rgeound::geound` function. If there is no improvement in the objective function in this number of generations, `rgenoud::genoud` will think that it has found the optimum.
`max`	logical. If `max=TRUE`, it indicates we wish to maximize the marginal quantile; if `max=FALSE`, we wish to minimize the marginal quantile. The default is `TRUE`.
`cl.setup`	the number of nodes. >1 indicates choosing parallel computing option in `rgenoud::genoud`. Default is 1.

`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug.)
`pop.size`	an integer with the default set to be 3000. This is the population number for the first generation in the genetic algorithm (`rgenoud::genoud`).
`hard_limit`	logical. When it is true the maximum number of generations in `rgeound::geound` cannot exceed 100. Otherwise, in this function, only `it.num` softly controls when genoud stops. Default is `FALSE`.

Details

Note that all estimation functions in this package use the same type of standardization on covariates. Doing so would allow us to provide a bounded domain of parameters for searching in the genetic algorithm.

For every stage `k`, $k = 1, 2$, this estimated parameters indexing the two-stage quantile-optimal treatment regime are returned *in two scales:*

1. , the returned `coef.k` is the set of parameters that we estimated after standardizing every covariate available for decision-making at stage `k` to be in the interval [0, 1]. To be exact, every covariate is subtracted by the smallest observed value and divided by the difference between the largest and the smallest value. Next, we carried out the algorithm in Wang et. al. 2017 to get the estimated regime parameters, `coef.k`, based on the standardized data. For the identifiability issue, we force the Euclidean norm of `coef.k` to be 1.
2. The difference between `coef.k` and `coef.orgn.scale.k` is that the latter set of parameters correspond to the original covariates, so the associated decision rule can be applied directly to novel observations. In other words, let β denote the estimated parameter in the original scale, then the estimated treatment regime is:

$$d(x) = I\{\beta_0 + \beta_1 x_1 + ... + \beta_k x_k > 0\},$$

   where the β values are returned as `coef.orgn.scale.k`, and the the vector $(1, x_1, ..., x_k)$ corresponds to the specified class of treatment regimes in the kth stage.

If, for every input covariate, the smallest observed value is exactly 0 and the range (i.e. the largest number minus the smallest number) is exactly 1, then the estimated `coef.k` and `coef.orgn.scale.k` will render identical.

Value

This function returns an object with 7 objects. Both `coefficients` and `coef.orgn.scale` were normalized to have unit euclidean norm.

`coef.1`, `coef.2` the set of parameters indexing the estimated quantile-optimal treatment regime for standardized covariates.

`coef.orgn.scale.1`, `coef.orgn.scale.2` the set of parameter indexing the estimated quantile-optimal treatment regime for the original input covariates.

`tau` the quantile of interest

`hatQ` the estimated marginal quantile when the treatment regime indexed by `coef.orgn.scale.1` and `coef.orgn.scale.2` is applied on the entire population. See the 'details' for connection between `coef.orgn.scale.k` and `coef.k`.

`call` the user's call.

`moPropen1`, `moPropen2` the user specified propensity score models for the first and the second stage respectively

`regimeClass.stg1`, `regimeClass.stg2` the user specified class of treatment regimes for the first and the second stage respectively

Author(s)

Yu Zhou, `<zhou0269@umn.edu>`

References

Wang L, Zhou Y, Song R and Sherwood B (2017). "Quantile-Optimal Treatment Regimes." *Journal of the American Statistical Association*.

Examples

```
ilogit <- function(x) exp(x)/(1 + exp(x))
GenerateData.2stg <- function(n){
 x1 <- runif(n)
 p1 <- ilogit(-0.5+x1)
 a1 <- rbinom(n, size=1, prob=p1)

 x2 <- runif(n,x1,x1+1)
 p2 <- ilogit(-1 + x2)
 a2 <- rbinom(n, size=1, prob=p2)

 mean <- 1+x1+a1*(1-3*(x1-0.2)^2) +x2 + a2*(1-x2-x1)
 y <- mean + (1+a1*(x1-0.5)+0.5*a2*(x2-1))*rnorm(n,0,sd = 1)
 return(data.frame(x1,a1,x2,a2,y))
}

n <- 400
testdata <- GenerateData.2stg(n)
fit <- TwoStg_Qopt(data=testdata, tau=0.2,
                   regimeClass.stg1=a1~x1, regimeClass.stg2=a2~x1+a1+x2,
                   moPropen1=a1~x1, moPropen2=a2 ~ x2,
                   cl.setup=2)
fit
```

Index