Package: RFQT (via r-universe)

September 26, 2024

Type Package

Title Random Forest of Q Trees

Version 0.1.0

Author Haodong Tian

Maintainer Haodong Tian <haodong.tian@mrc-bsu.cam.ac.uk>

Description Data-adaptive method for effect heterogeneity analysis or non-linear causal studies in Mendelian randomization and instrumental variables analysis.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Imports stats, MendelianRandomization, tidyverse, data.table, parallel

Suggests dplyr

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

RemoteUrl https://github.com/HDTian/RFQT

RemoteRef HEAD

RemoteSha 1edae49a2fcba3d1f64db12c1149947d1805469a

Contents

`BootstrapTreeFitting` *One Bootstrap Q Tree Fitting*

Description

`BootstrapTreeFitting` fits the Q-tree for a bootstrap data of the training set (with user-specific seed) and store the cleaned results. If you do not need results but the original (dirty) Q-tree information, you may use `GetTree()`.

Usage

```
BootstrapTreeFitting(seed=1,
                     Odat=odat,
                     Vdat=vdat,
                     honest=FALSE,
                     S=5,
                     rate=0.4,
                     SingleM=FALSE,
                     Qthreshold=3.84,
                     method='DR',
                     SoP=10,
                     howGX='SpecificGX',
                     Halve=FALSE,
                     endsize=1000,
                     const=NA)
```

Arguments

`seed`	seed value for reproducible and traceable results
`Odat`	a data frame. This is recognized as the trainging data set.
`Vdat`	a data frame. This is recognized as the testing or validation data set.
`honest`	Logical value indicates whether to use the honest estimation style or not. Default value is `FALSE`. When honest estimation is used, the tree construction and the leaf-specific estimates will be obtained in two seperate samples of the (boostrapped) training data.
`S`	a postive integer indicates the maximum tree depth allowed. Default value is 5 (i.e. the data set is allowed to be splitted at most 5 times). `S` is also conneted to the argument `endsize`, both of which work in a similar way. One may only restrict one of them and leave another one be a flexible value (e.g. `S=100` or `endsize=0`).
`rate`	a value ranging from 0 to 1 indicates the proportion of the candidate covariates to be randomly cosidered in each split when construcing the Q-tree.
`SingleM`	Logical value indicates whether to use one single covariate variable at all times. Default value is `FALSE`. If `SingleM=TRUE`, the covariate used are the best variable indicated by `GetIndex`.

`Qthreshold` A positive value used as the threshold value for the Q statistic. Default value is 3.0. One can set `Qthreshold=0` to ignore the Q-related stopping rule.

`method` a character indicates the stratification method used for constructing a Q-tree. There are currently three methods: the Doubly-ranked method (`method='DR'`), the residual methods (`method='Residual'`) and the naive method (recognized by any character except `'DR'` and `'Residual'`). The default method is `'DR'`.

`SoP` a postive integer (>=2) indicates the size of pre-straum. Only applicable for the doubly-ranked stratification. Default value is 10.

`howGX` a character indicates the way to calculate the instrument-exposure associations. Two ways are currenly allowed: the instrument-exposure associations estimated seperately at each stratum (`howGX='SpecificGX'`), or use a fixed instrument-exposure association (`howGX='const'`)

`Halve` Logical. Indictes whether to split the node by half:half (i.e. 5:5). Default value is `FALSE`.

`endsize` a positive integer value indicates the minimual size of the node of Q-tree. `S` and `endsize` work in a similar way.

`const` a value indicates the fixed instrument-exposure association value used for constructing the Q-tree. Only applicable for `howGX='SpecificGX'`.

Details

The training data `Odat` and the testing data `Vdat` should be of the same form and be a data frame that can be regonized by the functions. The first four columns must be the individual IDs, the (one-dimensional) instrument, the exposure (should be `NA` if not available), and the outcome, respectively. The (high-dimensional) covariates are then follwing. Note that all the functions may be working with incorrect column ordering, but the results are misleading. Do make sure the column variable order is correct.

If simulated data is used, there should be a end column named as true_STE. That is, `odat$true_STE` is not `NULL`.

If the data does not contain the exposure information, the RFQT can still be fitted, where one should use `howGX='const'` with a external instrument-exposure association (usually from independent studies).

Value

`BootstrapTreeFitting` returns a list with the following components:

`end_node_information`
the end nodes (leaves) information of the fitted Q tree, which contains the end-node index, the MR estimates, and the sample size proportion.

`OOB_predict` the predicted effects of the OOB samples accrding to the Q-tree. Values will be 0 if the individual is not out of bag.

`v_predict` the predicted effects of the testing samples accrding to the Q-tree.

`vi1`	the vector of the variable importance (VI) measurements for all the candidate covariates considered.
`vi2`	the vector of the variable importance (VI) measurements for all the candidate covariates considered. This VI measurements do not need the individual true effect information, therefore suitable for real application data.
`ts1`	the value of the permutation test with statistic S1.
`ts2`	the value of the permutation test with statistic S2.

When the real data is fitted, `vi1` is not applicable. When only traning data is available, `v_predict` is not applicable.

References

Burgess, S., Davies, N. M., & Thompson, S. G. (2014). "Instrumental variable analysis with a nonlinear exposure–outcome relationship". Epidemiology (Cambridge, Mass.), 25(6), 877. (Residual stratification)

Tian, H., Mason, A. M., Liu, C., & Burgess, S. (2023). "Relaxing parametric assumptions for non-linear Mendelian randomization using a doubly-ranked stratification method". PLOS Genetics, 19(6), e1010823. (Doubly-ranked stratification)

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the deflaut setting: scenario='A' and SoM=0.5
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set


#When running RFQT with mutiple Q trees/bootstrap - use parallel computation
Nb<-5 #how many trees in the forest? e.g. 5 trees
cl<-makeCluster(2)#    clusters/
clusterEvalQ(cl=cl , expr=library(dplyr))
clusterEvalQ(cl=cl , expr=library(MendelianRandomization) )
clusterExport(  cl=cl ,  varlist=c( 'odat', 'vdat',
                                    'GetTree', 'GetNindex', 'GetIndex' )  )#or any other arguments
RES<-parSapply(   cl ,  1:Nb, BootstrapTreeFitting  )
stopCluster(cl)
dim(RES)#7 Nb
##If you wish to use your own parameters rather than the default parameters, try:
#general exmaple
user_BootstrapTreeFitting<-function(seed){
  RES<-BootstrapTreeFitting(seed,
                            honest=my.honest,
                            S=my.S,
```

```
                              JJ=my.JJ,   #or any partial of the arguments
                              rate=my.rate,
                              Qthreshold=my.Qthreshold,
                              method=my.method,
                              SoP=my.SoP,
                              howGX=my.howGX,
                              endsize=my.endsize)
  return(RES)
}
#specific exmaple
user_BootstrapTreeFitting<-function(seed){
  RES<-BootstrapTreeFitting(seed,SoP=20)
  return(RES)
}
###parallel computation  -> RFQT
Nb<-5 #how many trees in the forest?
cl<-makeCluster(2)
clusterEvalQ(cl=cl , expr=library(dplyr))
clusterEvalQ(cl=cl , expr=library(MendelianRandomization) )
clusterExport(  cl=cl ,  varlist=c( 'odat', 'vdat',
                                    'GetTree', 'GetNindex', 'GetIndex' , 'BootstrapTreeFitting')  )
RES<-parSapply(   cl ,  1:Nb, user_BootstrapTreeFitting  )
stopCluster(cl)
```

`DTfit` *Fit a Classical Single Decision Tree (DT)*

Description

`DTfit` fits a data set (the training data) by the classical Q tree, and return the summary results. `DTfit` works similarly as `GetTree` and `BootstrapTreeFitting`, but not designed or used for future random forest fitting. If you wish to fit data via single tree rather than random forest, use `DTfit`.

Usage

```
DTfit(Odat = odat,
      Vdat = NA,
      honest = FALSE,
      S = 5,
      rate =1,
      SingleM = FALSE,
      Qthreshold = 3.84,
      method = "DR",
      SoP = 10,
      howGX = "SpecificGX",
      Halve = FALSE,
      endsize = 1000,
      const = NA)
```

Arguments

`Odat`	a data frame. This is recognized as the trainging data set.
`Vdat`	a data frame. This is recognized as the testing or validation data set. The default value is `NA`, indicating that no testing data will be considered.
`honest`	logical value indicates whether to use the honest estimation style or not. Default value is `FALSE`. When honest estimation is used, the tree construction and the leaf-specific estimates will be obtained in two seperate samples of the (boostrapped) training data.
`S`	a postive integer indicates the maximum tree depth allowed. Default value is 5 (i.e. the data set is allowed to be splitted at most 5 times). `S` is also conneted to the argument `endsize`, both of which work in a similar way. One may only restrict one of them and leave another one be a flexible value (e.g. `S=100` or `endsize=0`).
`rate`	a value ranging from 0 to 1 indicates the proportion of the candidate covariates to be randomly cosidered in each split when construcing the Q-tree.
`SingleM`	logical value indicates whether to use one single covariate variable at all times. Default value is `FALSE`. If `SingleM=TRUE`, the covariate used are the best variable indicated by `GetIndex`.
`Qthreshold`	a positive value used as the threshold value for the Q statistic. Default value is 3.84. One can set `Qthreshold=0` to ignore the Q-related stopping rule.
`method`	a character indicates the stratification method used for constructing a Q-tree. There are currently three methods: the Doubly-ranked method (`method='DR'`), the residual methods (`method='Residual'`) and the naive method (recognized by any character except `'DR'` and `'Residual'`). The default method is `'DR'`.
`SoP`	a postive integer (>=2) indicates the size of pre-straum. Only applicable for the doubly-ranked stratification. Default value is 10.
`howGX`	a character indicates the way to calculate the instrument-exposure associations. Two ways are currenly allowed: the instrument-exposure associations estimated seperately at each stratum (`howGX='SpecificGX'`), or use a fixed instrument-exposure association (`howGX='const'`).
`Halve`	logical. Indictes whether to split the node by half:half (i.e. 5:5). Default value is `FALSE`.
`endsize`	a positive integer value indicates the minimual size of the node of Q-tree. `S` and `endsize` work in a similar way.
`const`	a value indicates the fixed instrument-exposure association value used for constructing the Q-tree. Only applicable for `howGX='SpecificGX'`.

Details

Like `RFQTfit`, `DTfit` is an integrated function combining other functions, including `getPredict` and `getVI`.

The training data `Odat` and the testing data `Vdat` should be of the same form and be a data frame that can be regonized by the functions. The first four columns must be the individual IDs, the (one-dimensional) instrument, the exposure (should be `NA` if not available), and the outcome, respectively. The (high-dimensional) covariates are then follwing. Note that all the functions may be working with incorrect column ordering, but the results are misleading. DO make sure the column variable order is correct.

Value

`DTfit` returns a list consisting of the following components:

`end_node_information`
: the end nodes (leaves) information of the fitted Q tree, which contains the end-node index, the MR estimates, and the sample size proportion.

`v_predicted`
: the predicted effects of the testing samples accrding to the Q-tree.

`ts1`
: the value of the permutation test with statistic S1.

`ts2`
: the value of the permutation test with statistic S2.

`MSE`
: the MSE results for the testing samples (if both of the testing set and the label exist).

`rdat`
: the tree fitted data.

References

Burgess, S., Davies, N. M., & Thompson, S. G. (2014). "Instrumental variable analysis with a nonlinear exposure–outcome relationship". Epidemiology (Cambridge, Mass.), 25(6), 877. (Residual stratification)

Tian, H., Mason, A. M., Liu, C., & Burgess, S. (2023). "Relaxing parametric assumptions for non-linear Mendelian randomization using a doubly-ranked stratification method". PLOS Genetics, 19(6), e1010823. (Doubly-ranked stratification)

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the default setting: scenario='A' and SoM=0.5
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

DTRES<-DTfit(odat,vdat)

DTRES

#get (testing set) MSE manually
mean( ( DTRES$v_predict - vdat$true_STE   )^2   )
```

`getDat` *Creat a Simulated Data*

Description

`getDat` creats a toy data based on a certain model

Usage

```
getDat(N = 150000,
       Nt = 100000,
       Nc = 20,
       scenario='A',
       SoM = 0.5,
       ZXeffect = 0.5,
       Random = TRUE,
       label = TRUE,
       split = TRUE)
```

Arguments

`N`	a integer indicates the total sample size (training data size + testing data size)
`Nt`	a integer indicates the size of the training data.
`Nc`	a integer indicates the number of candidate variables.
`scenario`	a character indicates the model scenarios data will be simulated from. `'A'` refers to scenario where no colliders. `'B'` refers to the scenario that half covariates will be colliders. `'C'` refers to more complicated collider scenarios.
`SoM`	a value indicates the strength of modification (i.e. how strong will the covariate modify the treatment effect). Note even if `SoM=0`, there still exists weak modification.
`ZXeffect`	a value indicates the instrument-exposure effect.
`Random`	logical. Indicates whether to use the random value for strenght modification. Default value is `TRUE`, where weak modification always exists even if `SoM=0` due to randomness.
`label`	logical. Indicates whether to added the final column of the simulated data as the true heterogenerous effect (i.e. label). Default value is `TRUE`.
`split`	logical. If `FALSE`, only one data set is returened. The default is `TRUE`, the data will be splitted into the training data and testing data.

Details

The data-generating model is the same as the Scenarios in the original paper, where the first 5 covariates are the true effect modifiers. The covariates in even positions (2,4,...) are the common downstream variables of the exposure and te confounders, therefore causing collider bias (when conditioning on these variables).

Value

`getDat()` returns a list with three kings of toy data set

`whole.data`	the complete whole dataset
`traning.set`	the training set
`testing.set`	the testing set
`SoM`	Strength of Modification
`modifier_vec`	the modification strength vector for each covariate
`Scenario`	scenario information

the training set and the testing set are of the same form that can be regonized by all the functions. The first four columns are the individual IDs, the instrument, the exposure, and the outcome, respectively. The high-dimensional (dimensions = `Nc`) covariates are then follwing. The end column is `true_STE`, representing the individal controlled direct treatment effect (see more details in the original paper).

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)


set.seed(60)
res<-getDat() #simulated data #the deflaut setting: scenario='A' and SoM=0.5
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set
```

GetIndex *Get the Best Covariate Index*

Description

`GetIndex` gives the best covariate index according to the inputed data set. The best covariate is one out of the candidate covariates considered that gives the largest Q statistic value.

Usage

```
GetIndex(dat_current,
          JJ,
          rate = 1,
          SpecificM = NA,
          method = "DR",
          SoP = 10,
          howGX = "SpecificGX",
          Halve = FALSE,
          const = NA)
```

Arguments

dat_current	A data frame. This data frame has the first four columns: individual IDs, the instrument, the expoure, and the outcome with the candidate covariates following.
JJ	A positive integer indicates the total number of the candiate covariates.
rate	a value ranging from 0 to 1 indicates the proportion of the candidate covariates to be randomly cosidered in each split when construcing the Q-tree.
SpecificM	a vector indicates the index of candidate variables that can be considered in spliiting. Tbe default value is NA, which means all candidate variables are possible to be considered.
method	a character indicates the stratification method used for constructing a Q-tree. There are currently three methods: the Doubly-ranked method (method='DR'), the residual methods (method='Residual') and the naive method (recognized by any character except 'DR' and 'Residual'). The default method is 'DR'.
SoP	a postive integer (>=2) indicates the size of pre-straum. Only applicable for the doubly-ranked stratification. Default value is 10.
howGX	a character indicates the way to calculate the instrument-exposure associations. Two ways are currenly allowed: the instrument-exposure associations estimated seperately at each stratum (howGX='SpecificGX'), or use a fixed instrument-exposure association (howGX='const')
Halve	Logical. Indictes whether to split the node by half:half (i.e. 5:5). Default value is FALSE.
const	a value indicates the fixed instrument-exposure association value used for constructing the Q-tree. Only applicable for howGX='SpecificGX'.

Details

GetIndex helps to decide the splitting covariate at the present node when constructing a Q-tree.

Value

GetIndex returns a vector with the following three values:'Candidate.index', 'Q.value' , 'node.size'

Candidate.index	The index (position rank) of the chosen covariate.
Q.value	The corresponding Q statistic value of the chosen covariate.
minimal.node.size.after.split	The mimimal sample size of the sun-nodes after splitting.
split.style	Three values denoted by 1,2,3 representing the splitting style used for the final covariate. They are 3:7, 5:5, 7:3, respectively.

References

Burgess, S., Davies, N. M., & Thompson, S. G. (2014). "Instrumental variable analysis with a nonlinear exposure–outcome relationship". Epidemiology (Cambridge, Mass.), 25(6), 877. (Residual stratification)

Tian, H., Mason, A. M., Liu, C., & Burgess, S. (2022). "Relaxing parametric assumptions for non-linear Mendelian randomization using a doubly-ranked stratification method". bioRxiv, 2022-06. (Doubly-ranked stratification)

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data #the deflaut setting: scenario='A' and SoM=0.5
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set
GetIndex(odat,JJ=20)
```

`getMSE` *Get MSE Values for the OOB Data or the Testing Data with Fitted Q-Tree or Random Forest of Q-Trees (RFQT)*

Description

`getMSE` calculates the MSE value for RFQT (or a Q-tree if RFQT only contains one single tree)

Usage

```
getMSE(RES,
       indicator = 1
       )
```

Arguments

`RES`	a fitting list result from `BootstrapTreeFitting` or `RFQTfit(odat,vdat)$RES`.
`indicator`	numeric indicator to judge which type of MSE should be calculated. `indicator=1` for Out-of-Bag (OOB) data and `indicator=2` for th testing data. Defalut value is `indicator=1`.

Value

`getMSE` returns a vector of MSE with the increase of the number of Q-trees. The stable MSE values should indicate an appropriate number of Q-trees.

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the deflaut setting: scenario='A' and SoM=0.5
res<-getDat(label=FALSE)
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

##When running RFQT with mutiple Q trees/bootstrap - use parallel computation
Nb<-7 #how many trees in the forest? e.g. 5 trees
cl<-makeCluster(2)
clusterEvalQ(cl=cl , expr=library(dplyr))
clusterEvalQ(cl=cl , expr=library(MendelianRandomization) )
clusterExport(  cl=cl ,  varlist=c('odat','vdat',
                                   'GetTree', 'GetNindex', 'GetIndex' )  )
RES<-parSapply(   cl ,  1:Nb, BootstrapTreeFitting  )
stopCluster(cl)
dim(RES)#7 Nb


getMSE(RES,indicator=1 )
getMSE(RES,indicator=2 )
```

`GetNindex` *Get the Q Tree End Node Index for Choosen Samples*

Description

`GetNindex` helps to match the Q-tree end node for any inputed samples, and therefore enables to make further analysis like effect prediction for these inputed samples.

Usage

```
GetNindex(M,
          rdat,
          S = NA)
```

Arguments

`M`	A matrix or data frame contains the candidate covariate information.
`rdat`	A data frame. It is the Q-tree informaton result and obtained by `rdat<-GetTree()`.
`S`	A positive integer indicates the max depth of each sample will explore along the tree. This number should be consistent with the max depth of the fitted Q tree (also refected in `rdat`). The default valus if `NA`, which means the code will automatically use the correct `S` value.

Details

The inputed data set `M` must contain the same candidate covariate information (with the same order) as the training data for the Q-tree. In addition, the first four columns (individual IDs, the instrument, the exposure, the outcome) need to be removed, so that the fist column starts from the first candidate covariate.

Value

`GetNindex` returns a vector with the same length of the row number of the inputed data `M`. Each element represents the end node (i.e. leaf) index according to the reference Q-tree (the tree information was stored within `rdat`).

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data #the deflaut setting: scenario='A' and SoM=0.5
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

rdat<-GetTree(odat)


vdat_Nindex<-GetNindex(vdat[,5:24] ,rdat )
#the M colnumber and order should be same as the training data


#may use another independent data (estimation data) to
#calculate the endnode-specific IV/MR estimates

#for example:
#let odat halves into trdata (tree data) and estdata (estimation data);
#also we can have rdat<-GetTree(trdata)

trdat<-odat[1: (nrow(odat)/2) ,]#tree data
estdat<-odat[(nrow(odat)/2+1):nrow(odat) ,] #estimaiton data

rdat<-GetTree(trdat)#fitted Q tree

estdat$Nindex<-GetNindex(estdat[,5:24] ,rdat )
#rdat contains the tree information (i.e. decision rule)
```

---

getPars *Get the present parameters information*

Description

`getPars` returns the information of the (hyper-)parameters relevant to the Q-tree and RFQT fitting.

Usage

```
getPars(empty.argument)
```

Arguments

`empty.argument`
: No arguments are needed.

Value

`getPars()` returns the following information: the total number of candidate covariates, the training data size, the testing data siz, the stratification method, the size of pre-stratum (if applicable), the proportion of the covariates randomly considered in each node split, the maximal tree depth allowed, the way to calculate the instrument-exposure associations, the fixed instrument-exposure association level (if applicable), the minimal node size, the threshold value of the Q statistic.

If the variable is not defined, it will return 'Not defined' with a value in brackets representing the default value for this variable when fitting RFQT.

Note that only the variable defined in the global enviroment will be considered. Even if one defines these variables, the functions like `BootstrapTreeFitting` and `RFQTfit` will use the default value if (s)he does not modify the function arguments.

See the original paper for the details of each parameter information.

Author(s)

Haodong Tian

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

res<-getDat()
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

method<-'DR'; SoP<-20; rate<-2/5

getPars()
```

`getPredict` *Get predict effects for the OOB Data or the Testing Data with Fitted Random Forest of Q-Trees (RFQT)*

Description

`getPredict` calculates the predicted individual treatment effect estimates for RFQT, either for the OOB data or the testing data.

Usage

```
getPredict(RES,
           indicator = 1
           )
```

Arguments

`RES`	a fitting list result from `BootstrapTreeFitting` or `RFQTfit(odat,vdat)$RES`.
`indicator`	numeric indicator to judge which type of predicted effect should be presented. `indicator=1` for Out-of-Bag (OOB) data and `indicator=2` for th testing data. Defalut value is `indicator=1`.

Details

See the relevant part of the original paper for more details of how to predict an individual effect according to a fitted Q tree (therefore a RFQT).

Value

`getPredict` returns a matrix where the row corresponds to the sample size (either the OOB data or the testing data, depending on the indicator used) and the column coresponds to the number of Q-trees. Each column represents the individual predicted effects with the present number of Q trees.

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the deflaut setting: scenario='A' and SoM=0.5
res<-getDat(label=FALSE)
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

##When running RFQT with mutiple Q trees/bootstrap - use parallel computation
Nb<-7 #how many trees in the forest? e.g. 5 trees
```

```
cl<-makeCluster(2)
clusterEvalQ(cl=cl , expr=library(dplyr))
clusterEvalQ(cl=cl , expr=library(MendelianRandomization) )
clusterExport(  cl=cl ,  varlist=c('odat','vdat',
                                   'GetTree', 'GetNindex', 'GetIndex' )  )
RES<-parSapply(   cl ,  1:Nb, BootstrapTreeFitting  )
stopCluster(cl)
dim(RES)#7 Nb

predict_matrix<-getPredict(RES,indicator=1 )
predict_matrix<-getPredict(RES,indicator=2 )

dim(predict_matrix)
###View(predict_matrix)
```

`GetTree` *Fit and Get a Q Tree*

Description

`GetTree` fits a data and construct a Q-tree. The fitted data is directly used to build tree (so no bootstrap). The tree information will be stored but not cleaned for further analysis. You may need `BootstrapTreeFitting` rather than this function.

Usage

```
GetTree(dat,
        S = 5,
        Qthreshold = 3.84,
        rate = 1,
        SpecificM = NA,
        method = "DR",
        SoP = 10,
        howGX = "SpecificGX",
        Halve = FALSE,
        const = NA,
        endsize = 1000)
```

Arguments

`dat`	a data frame. This data (without any bootstrap) will be fitted to construct a Q-tree.
`S`	a postive integer indicates the maximum tree depth allowed. Default value is 5 (i.e. the data set is allowed to be splitted at most 5 times). S is also conneted to the argument endsize, both of which work in a similar way. One may only restrict one of them and leave another one be a flexible value (e.g. `S=100` or `endsize=0`).

`Qthreshold` A positive value used as the threshold value for the Q statistic. Default value is 3.0.

`rate` a value ranging from 0 to 1 indicates the proportion of the candidate covariates to be randomly cosidered in each split when constrcuing the Q-tree. When `rate=1`, it means all the candiate covariates will be considered.

`SpecificM` a vector indicates the index of candidate variables that can be considered in spliiting. Tbe default value is `NA`, which means all candidate variables are possible to be considered.

`method` a character indicates the stratification method used for constructing a Q-tree. There are currently three methods: the Doubly-ranked method (`method='DR'`), the residual methods (`method='Residual'`) and the naive method (recognized by any character except `'DR'` and `'Residual'`). The default method is `'DR'`.

`SoP` a postive integer (>=2) indicates the size of pre-straum. Only applicable for the doubly-ranked stratification. Default value is 10.

`howGX` a character indicates the way to calculate the instrument-exposure associations. Two ways are currenly allowed: the instrument-exposure associations estimated seperately at each stratum (`howGX='SpecificGX'`), or use a fixed instrument-exposure association (`howGX='const'`)

`Halve` Logical. Indictes whether to split the node by half:half (i.e. 5:5). Default value is `FALSE`.

`const` a value indicates the fixed instrument-exposure association value used for constructing the Q-tree. Only applicable for `howGX='SpecificGX'`.

`endsize` a positive integer value indicates the minimual size of the node of Q-tree allowed. One can also control the endsize via `S`.

Details

`dat` must be a data frame that can be regonized by the functions. The first four columns must be the individual IDs, the (one-dimensional) instrument, the exposure (should be `NA` if not available), and the outcome, respectively. The (high-dimensional) covariates are then follwing. Note that all the functions may be working with incorrect column ordering, but the results are misleading. DO make sure the column variable order is correct.

If simulated data is used, there should be a end column named as true_STE. That is, `dat$true_STE` is not `NULL`.

Value

`GetTree` returns a data set with the same row dimension as the inputed data `dat`. The returned data is 'augmented' with tree information including the tree end node index, and the splitting information in each split.

References

Burgess, S., Davies, N. M., & Thompson, S. G. (2014). "Instrumental variable analysis with a nonlinear exposure–outcome relationship". Epidemiology (Cambridge, Mass.), 25(6), 877. (Residual stratification)

Tian, H., Mason, A. M., Liu, C., & Burgess, S. (2023). "Relaxing parametric assumptions for non-linear Mendelian randomization using a doubly-ranked stratification method". PLOS Genetics, 19(6), e1010823. (Doubly-ranked stratification)

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

res<-getDat()
odat<-res$traning.set  #training set

tree_result<-GetTree(odat)
```

---

getVI — *Get Varibale Importance Measurements with Fitted Q-Tree or Random Forest of Q-Trees (RFQT)*

Description

`getVI` calculates the Varibale Importance (VI) measurements for a random forest of Q-trees (RFQT).

Usage

```
getVI(RES,
      VItype = 2
      )
```

Arguments

`RES`	a fitting list result from `BootstrapTreeFitting` or `RFQTfit(odat,vdat)$RES`.
`VItype`	numeric indicator to judge which type of VI measurement should be calculated. `VItype=1` for the classical way with the known individual treatment effects (i.e. label) and `VItype=2` for case that the individual treatment effects are unavailable (as in most real application). Defalut value is `VItype=2`.

Details

See the VI measurement part of the original paper for more detailed VI introduction and the algorithm.

Value

`getVI` retuns a vector indicating the variable importacne (VI) measurements for all the candidate covariates (with the same ordering of the candidate covariates to the original fitting data).

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the deflaut setting: scenario='A' and SoM=0.5
res<-getDat(label=FALSE)
odat<-res$traning.set  #training set
vdat<-res$testing.set  #testing set

##When running RFQT with mutiple Q trees/bootstrap - use parallel computation
Nb<-7 #how many trees in the forest? e.g. 5 trees
cl<-makeCluster(2)
clusterEvalQ(cl=cl , expr=library(dplyr))
clusterEvalQ(cl=cl , expr=library(MendelianRandomization) )
clusterExport(  cl=cl ,  varlist=c('odat','vdat',
                                   'GetTree', 'GetNindex', 'GetIndex' )  )
RES<-parSapply(   cl ,  1:Nb, BootstrapTreeFitting  )
stopCluster(cl)
dim(RES)#7 Nb

getVI(RES,VItype=1 )
getVI(RES,VItype=2 )
```

`RFQTfit` *Fit a Random Forest of Q Trees (RFQT)*

Description

`RFQTfit` fits a data set (the training data) by the random forest of Q trees (RFQT), and return the summary results.

Usage

```
RFQTfit(odat,
        vdat = NA,
        Nb = 5,
        S = 5,
        honest = FALSE,
        rate = 0.4,
        SingleM = FALSE,
        Qthreshold = 3.84,
        method = "DR",
        SoP = 10,
        howGX = "SpecificGX",
        Halve = FALSE,
```

```
endsize = 1000,
const = NA,
Cores = NA,
trackfile=NA
)
```

Arguments

`odat`	a data frame. This is recognized as the trainging data set.
`vdat`	a data frame. This is recognized as the testing or validation data set. The default value is `NA`, indicating that no testing data will be considered.
`Nb`	a positive integer indicates the number of bootstrap (i.e. the number of Q-trees)
`S`	a postive integer indicates the maximum tree depth allowed. Default value is 5 (i.e. the data set is allowed to be splitted at most 5 times). `S` is also conneted to the argument `endsize`, both of which work in a similar way. One may only restrict one of them and leave another one be a flexible value (e.g. `S=100` or `endsize=0`).
`honest`	logical value indicates whether to use the honest estimation style or not. Default value is `FALSE`. When honest estimation is used, the tree construction and the leaf-specific estimates will be obtained in two seperate samples of the (boostrapped) training data.
`rate`	a value ranging from 0 to 1 indicates the proportion of the candidate covariates to be randomly cosidered in each split when construcing the Q-tree.
`SingleM`	logical value indicates whether to use one single covariate variable at all times. Default value is `FALSE`. If `SingleM=TRUE`, the covariate used are the best variable indicated by `GetIndex`.
`Qthreshold`	a positive value used as the threshold value for the Q statistic. Default value is 3.84. One can set `Qthreshold=0` to ignore the Q-related stopping rule.
`method`	a character indicates the stratification method used for constructing a Q-tree. There are currently three methods: the Doubly-ranked method (`method='DR'`), the residual methods (`method='Residual'`) and the naive method (recognized by any character except `'DR'` and `'Residual'`). The default method is `'DR'`.
`SoP`	a postive integer (>=2) indicates the size of pre-straum. Only applicable for the doubly-ranked stratification. Default value is 10.
`howGX`	a character indicates the way to calculate the instrument-exposure associations. Two ways are currenly allowed: the instrument-exposure associations estimated seperately at each stratum (`howGX='SpecificGX'`), or use a fixed instrument-exposure association (`howGX='const'`).
`Halve`	logical. Indictes whether to split the node by half:half (i.e. 5:5). Default value is `FALSE`.
`endsize`	a positive integer value indicates the minimual size of the node of Q-tree. `S` and `endsize` work in a similar way.

`const` a value indicates the fixed instrument-exposure association value used for constructing the Q-tree. Only applicable for `howGX='SpecificGX'`.

`Cores` a positive integer indicates the number of cores should be used when running random forest in parallel. Default value is the number of CPU cores minus one. You can check the number of CPU cores via `detectCores()`.

`trackfile` the path string indicates the file to store the running information in parallel.

Details

`RFQTfit` is an integrated function combining other functions, including `GetTree`, `BootstrapTreeFitting`, `getPredict`, `getVI`, etc. The fitting automatically uses parallel computation.

The training data `Odat` and the testing data `Vdat` should be of the same form and be a data frame that can be regonized by the functions. The first four columns must be the individual IDs, the (one-dimensional) instrument, the exposure (should be `NA` if not available), and the outcome, respectively. The (high-dimensional) covariates are then follwing. Note that all the functions may be working with incorrect column ordering, but the results are misleading. DO make sure the column variable order is correct.

Value

`RFQTfit` shows all the parameter information used for random forest fitting in the begining, and returns in the end a list consisting of the following components:

`RES` The direct fitting results of each Q-tree, which contains `end_node_information`, `OOB_predict`, `v_predict`, `vi1`, `vi2`, `ts1` and `ts2` (same as the results of `BootstrapTreeFitting`) for each Q-tree. Note that `RES` is multi-dimensional. If one wishes to obtain the individual predicted effects of the testing set according to i-th Q-tree, should run `RES[3,i]$v_predict`.

`MSE_OOB` the `Nb`-length vector of MSE results for OOB samples. If one determine the number of Q-trees to be `Nb`, simply use the final vlue of `MSE_OOB` as the MSE.

`MSE_test` the `Nb`-length vector of MSE results for test samples (if the testing set exists). If one determine the number of Q-trees to be `Nb`, simply use the final vlue of `MSE_test` as the MSE.

`Predict_OOB` the matrix of predicted effects for OOB samples, where the row corresponds to the sample size (the testing data) and the column coresponds to the number of Q-trees. Each column represents the individual predicted effects with the present number of Q trees. If one determine the number of Q-trees to be `Nb`, simply use the final colunm of `Predict_OOB` as the individual predicted effect vector.

`Predict_test` the matrix of predicted effects for testing set, where the row corresponds to the sample size (the testing data) and the column coresponds to the number of Q-trees. Each column represents the individual predicted effects with the present number of Q trees. If one determine the number of Q-trees to be `Nb`, simply use the final colunm of `Predict_test` as the individual predicted effect vector.

`VI1`	the vector indicates the type I (i.e. with labels) variable importacne (VI) measurements for all the candidate covariates (with the same ordering of the candidate covariates to the original fitting data).
`VI2`	the vector indicates the type II (i.e. without labels) variable importacne (VI) measurements for all the candidate covariates (with the same ordering of the candidate covariates to the original fitting data).
`TS`	the vector of the permutation test statistic values. The two values represent the values of TS1 and TS2, respectively.

References

Burgess, S., Davies, N. M., & Thompson, S. G. (2014). "Instrumental variable analysis with a nonlinear exposure–outcome relationship". Epidemiology (Cambridge, Mass.), 25(6), 877. (Residual stratification)

Tian, H., Mason, A. M., Liu, C., & Burgess, S. (2023). "Relaxing parametric assumptions for non-linear Mendelian randomization using a doubly-ranked stratification method". PLOS Genetics, 19(6), e1010823. (Doubly-ranked stratification)

Examples

```
library(MendelianRandomization)
library(tidyverse)
library(data.table)
library(parallel)

set.seed(60)
res<-getDat() #simulated data  #the deflaut setting: scenario='A' and SoM=0.5
odat<-res$traning.set
vdat<-res$testing.set

###try: ALLRES<-RFQTfit(odat)

###try: ALLRES<-RFQTfit(odat,vdat,Nb=200)
```

Index