Package 'MultiObjMatch'

November 23, 2023

Type Package

Title Multi-Objective Matching Algorithm

Version 0.1.3

Description Matching algorithm based on network-flow structure.
Users are able to modify the emphasis on three different
optimization goals: two different distance measures and
the number of treated units left unmatched. The method is proposed by Pimentel
and Kelz (2019) <doi:10.1080/01621459.2020.1720693>.
The 'rrelaxiv' package, which provides an alternative solver for
the underlying network flow problems, carries an
academic license and is not available on CRAN, but
may be downloaded from Github at
<`https://github.com/josherrickson/rrelaxiv/`>.

License MIT + file LICENSE

Encoding UTF-8

Imports cobalt, dplyr, optmatch, matchMulti, fields, plyr, RCurl,
gtools, rcbalance, MASS, stats, methods, utils, rlemon

RoxygenNote 7.1.2

Suggests testthat (>= 3.0.0), rrelaxiv

Additional_repositories `https://errickson.net/rrelaxiv/`

Config/testthat/edition 3

NeedsCompilation no

Author Shichao Han [cre, aut],
Samuel D. Pimentel [aut]

Maintainer Shichao Han <`schan21@berkeley.edu`>

Repository CRAN

Date/Publication 2023-11-23 08:00:05 UTC

R topics documented:

`addBalance` *Add fine balance edges*

Description

Add fine balance edges

Usage

```
addBalance(net, treatedVals, controlVals, replaceExisting = TRUE)
```

Arguments

`net`	the network object created for the network flow problem
`treatedVals`	the balance value for treated nodes
`controlVals`	the balance value for control nodes
`replaceExisting`	(optional) whether or not to replace the existing net; TRUE by default

Value

the network structure with balance edges added

`addExclusion` *Add exclusion edges*

Description

Add exclusion edges

Usage

```
addExclusion(net, remove = FALSE)
```

Arguments

`net`	the input network structure
`remove`	(optional) whether to exclude edges; FALSE by default

Value

the network structure with exclusion edges added to allow for trdeoff for the exclusion cost

---

`balanceCosts` *Create a skeleton representation of the balance edge costs*

Description

Create a skeleton representation of the balance edge costs associated with pairings for a given distance and network

Usage

```
balanceCosts(net, balance.penalty = 1)
```

Arguments

`net`	the network structure
`balance.penalty`	(optional) the numeric value for balance; 1 by default

Value

the skeleton with balance edge cost

`build.dist.struct` *An internal helper function that generates the data abstraction for the edge weights of the main network structure.*

Description

An internal helper function that generates the data abstraction for the edge weights of the main network structure.

Usage

```
build.dist.struct(
  z,
  X,
  distMat,
  exact = NULL,
  dist.type = "Mahalanobis",
  calip.option = "propensity",
  calip.cov = NULL,
  caliper = 0.2,
  verbose = FALSE
)
```

Arguments

`z`	a vector of treatment and control indicators, 1 for treatment and 0 for control.
`X`	a data frame or a numeric or logical matrix containing covariate information for treated and control units. Its row count must be equal to the length of z.
`distMat`	a matrix of pair-wise distance specified by the user
`exact`	an optional vector of the same length as z. If this argument is specified, treated units will only be allowed to match to control units that have equal values in the corresponding indices of the exact vector. For example, to match patients within hospitals only, one could set exact equal to a vector of hospital IDs for each patient.
`dist.type`	one of ('propensity','user','none'). If 'propensity' is specified (the default option), the function estimates a propensity score via logistic regression of z on X and imposes a propensity score caliper. If 'user' is specified, the user must provide a vector of values on which a caliper will be enforced using the calip.cov argument. If 'none' is specified no caliper is used.
`calip.option`	a character indicating the type of caliper used
`calip.cov`	see calip.option.
`caliper`	a numeric value that gives the size of the caliper when the user specifies the calip.option argument as 'propensity' or 'calip.cov'.
`verbose`	a boolean value whether to print(cat) debug information. Default: FALSE

Value

a distance structure used for constructing the main network flow problem

---

`build.dist.struct_user`

An internal helper function that generates the data abstraction for the edge weights of the main network structure using the distance matrix passed by the user.

Description

An internal helper function that generates the data abstraction for the edge weights of the main network structure using the distance matrix passed by the user.

Usage

```
build.dist.struct_user(z, distMat, verbose = FALSE)
```

Arguments

`z`	a vector indicating whether each unit is in treatment or control group
`distMat`	a matrix of pair-wise distance
`verbose`	a boolean value whether to print(cat) debug information. Default: FALSE

Value

a distance structure used for constructing the main network flow problem

---

`callrelax` *Call relax on the network*

Description

this function is copied from the rcbalance package

Usage

```
callrelax(net, solver = "rlemon")
```

Arguments

`net`	the network structure
`solver`	(optional) the solver; by default, "rlemon"

Value

list of the result from the call to relax solver

`combine_dist` *An internal helper function that combines two distance object*

Description

An internal helper function that combines two distance object

Usage

```
combine_dist(a, b)
```

Arguments

`a`	a distance structure object
`b`	a distance structure object

Value

a new distance structure object whose edge weights are the sum of the corresponding edge weigths in a and b compareMatching *Generate covariate balance in different matches*

Description

This is a wrapper function for use in evaluating covariate balance across different matches. The function calls `compareTables` on the output from the function `generateBalanceTable`. It only works for 'Basic' version of matching (using `distBalMatch`).

Usage

```
compareMatching(
  matchingResult,
  covList = NULL,
  display.all = TRUE,
  stat = "mean.diff"
)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch
`covList`	(optional) factor of names of covariates that we want to evaluate covariate balance on; default is NULL. When set to NULL, the program will compare the covariates that have been used to construct a propensity model.
`display.all`	(optional) boolean value of whether to display all the matches; default is TRUE, where matches at each quantile is displayed
`stat`	(optional) character of the name of the statistic used for measuring covariate balance; default is "mean.diff". This argument is the same as used in "cobalt" package, see: bal.tab

Value

a dataframe that shows covariate balance in different matches

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Generate table for comparing matches
compareMatching(matchResult, display.all = TRUE)
```

compareTables *Summarize covariate balance table*

Description

This function would take the result of `generateBalanceTable` function and combine the results in a single table. It only works for 'Basic' version of the matching.

Usage

```
compareTables(balanceTable)
```

Arguments

`balanceTable`	a named list, which is the result from the function `generateBalanceTable`

Value

a dataframe with combined information

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Generate summary table for comparing matches
compareTables(generateBalanceTable(matchResult))
```

`convert_index` *An internal helper function that translates the matching index in the sorted data frame to the original dataframe's row index*

Description

An internal helper function that translates the matching index in the sorted data frame to the original dataframe's row index

Usage

```
convert_index(matchingResult)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch

---

`convert_names` *Internal helper function that converts axis name to internal variable name*

Description

Internal helper function that converts axis name to internal variable name

Usage

```
convert_names(x, y)
```

Arguments

`x`	the user input character for x-axis value
`y`	the user input character for y-axis value

Value

a named list with variable names for visualization for internal use

---

`costSkeleton` *Create cost skeleton*

Description

Create a more user-friendly data structure to represent the edge costs in a network. Internally the network object used by the optmiization routine represents all the edge costs in a single vector. The "skeleton" structure decomposes this vector into a list of components, each corresponding to a different role in the network: "pairings" are edges between treated and control, exclusion" are direct links between treated units and a sink that allows them to be excluded, "balance" refers to edges that count marginal balance between groups, and "sink" indicates edges that connect control nodes to the sink. Skeletons are created so these various features can be combined (or switched on and off) easily into objective functions, and the interface to the main tradeoff function expects to see each function represented in skeleton format.

Usage

```
costSkeleton(net)
```

Arguments

`net`	the network structure

Value

the skeleton

`data_precheck` *Data precheck: Handle missing data(mean imputation) and remove redundant columns; it also adds an NA column for indicating whether it's missing*

Description

Data precheck: Handle missing data(mean imputation) and remove redundant columns; it also adds an NA column for indicating whether it's missing

Usage

```
data_precheck(X)
```

Arguments

`X`	a dataframe that the user initially inputs for matching - dataframe with covariates

Value

a dataframe with modified data if necessary

`descr.stats_general` *Generate summary statistics for matches*

Description

Generate summary statistics for matches

Usage

```
descr.stats_general(matches, df, treatCol, b.vars, pair.vars, extra = FALSE)
```

Arguments

`matches`	One matching result from the main matching function
`df`	the original data frame used for matching
`treatCol`	the character of the column name for treatment vector
`b.vars`	the vector of column names of covariates used for measuring balance
`pair.vars`	the vector of column names used for measuring pairwise distance
`extra`	the list of summary statistic; it must be the types that can be taken by cobalt

Value

a named vector of summary statistic

`distanceFunctionHelper` *Helper function that change input distance matrix*

Description

Helper function that change input distance matrix

Usage

```
distanceFunctionHelper(z, distMat)
```

Arguments

`z`	the treatment vector
`distMat`	the user input distance matrix

Value

a distance matrix where (i,j) element is the distance between unit i and j in the same order as z

distBalMatch *Optimal tradeoffs among distance, exclusion and marginal imbalance*

Description

Explores tradeoffs among three important objective functions in an optimal matching problem:the sum of covariate distances within matched pairs, the number of treated units included in the match, and the marginal imbalance on pre-specified covariates (in total variation distance).

Usage

```
distBalMatch(
  df,
  treatCol,
  myBalCol,
  rhoExclude = c(1),
  rhoBalance = c(1, 2, 3),
  distMatrix = NULL,
  distList = NULL,
  exactlist = NULL,
  propensityCols = NULL,
  pScores = NULL,
  ignore = NULL,
  maxUnMatched = 0.25,
  caliperOption = NULL,
  toleranceOption = 0.01,
  maxIter = 0,
  rho.max.f = 10
)
```

Arguments

`df`	data frame that contain columns indicating treatment, outcome and covariates.
`treatCol`	character of name of the column indicating treatment assignment.
`myBalCol`	character of column name of the variable on which to evaluate marginal balance.
`rhoExclude`	(optional) numeric vector of values of exclusion penalty. Default value is c(1).
`rhoBalance`	(optional) factor of values of marginal balance penalty. Default value is c(1,2,3).
`distMatrix`	(optional) a matrix that specifies the pair-wise distances between any two objects.
`distList`	(optional) character vector of variable names used for calculating within-pair distance.
`exactlist`	(optional) character vector, variable names that we want exact matching on; NULL by default.
`propensityCols`	(optional) character vector, variable names on which to fit a propensity score (to supply a caliper).

`pScores`	(optional) character, giving the variable name for the fitted propensity score.
`ignore`	(optional) character vector of variable names that should be ignored when constructing the internal matching. NULL by default.
`maxUnMatched`	(optional) numeric, the maximum proportion of unmatched units that can be accepted; default is 0.25.
`caliperOption`	(optional) numeric, the propensity score caliper value in standard deviations of the estimated propensity scores; default is NULL, which is no caliper.
`toleranceOption`	(optional) numeric, tolerance of close match distance; default is 1e-2.
`maxIter`	(optional) integer, maximum number of iterations to use in searching for penalty combintions that improve the matching; default is 0.
`rho.max.f`	(optional) numeric, the scaling factor used in proposal for rhos; default is 10.

Details

Matched designs generated by this function are Pareto optimal for the three objective functions. The degree of relative emphasis among the three objectives in any specific solution is controlled by the penalties, denoted by Greek letter rho. Larger values of `rhoExclude` corresponds to increased emphasis on retaining treated units (all else being equal), while larger values of `rhoBalance` corresponds to increased emphasis on marginal balance. Additional details:

- Users may either specify their own distance matrix via the `distMatrix` argument or ask the function to create a Mahalanobis distance matrix internally on a set of covariates specified by the `distList` argument; if neither argument is specified an error will result. User-specified distance matrices should have row count equal to the number of treated units and column count equal to the number of controls.
- If the `caliperOption` argument is specified, a propensity score caliper will be imposed, forbidding matches between units more than a fixed distance apart on the propensity score. The caliper will be based either on a user-fit propensity score, identified in the input dataframe by argument `pScores`, or by an internally-fit propensity score based on logistic regression against the variables named in `psoreCols`. If `caliperOption` is non-NULL and neither of the other arguments is specified an error will result.
- `toleranceOption` controls the precision at which the objective functions is evaluated. When matching problems are especially large or complex it may be necessary to increase toleranceOption in order to prevent integer overflows in the underlying network flow solver; generally this will be suggested in appropariate warning messages.
- While by default tradeoffs are only assessed at penalty combinations provided by the user, the user may ask for the algorithm to search over additional penalty values in order to identify additional Pareto optimal solutions. `rho.max.f` is a multiplier applied to initial penalty values to discover new solutions, and setting it larger leads to wider exploration; similarly, `maxIter` controls how long the exploration routine runs, with larger values leading to more exploration.

Value

a named list whose elements are: * "rhoList": list of penalty combinations for each match * "matchList": list of matches indexed by number

- "treatmentCol": character of treatment variable
- "covs": character vector of names of the variables used for calculating within-pair distance
- "exactCovs": character vector of names of variables that we want exact or close match on * "idMapping": numeric vector of row indices for each observation in the sorted data frame for internal use
- "stats": data frame of important statistics (total variation distance) for variable on which marginal balance is measured
- "b.var": character, name of variable on which marginal balance is measured * "dataTable": data frame sorted by treatment value
- "t": a treatment vector
- "df": the original dataframe input by the user
- "pair_cost1": list of pair-wise distance sum using the first distance measure
- "pair_cost2": list of pair-wise distance sum using the second distance measure (left NULL since only one distance measure is used here).
- "version": (for internal use) the version of the matching function called; "Basic" indicates the matching comes from distBalMatch and "Advanced" from twoDistMatch.
- "fPair": a vector of values for the first objective function; it corresponds to the pair-wise distance sum according to the first distance measure.
- "fExclude": a vector of values for the second objective function; it corresponds to the number of treated units being unmatched.
- "fMarginal": a vector of values for the third objective function; it corresponds to the marginal balanced distance for the specified variable(s).

See Also

Other main matching function: `twoDistMatch()`

Examples

```
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree", "race")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree", "race")
myBalVal <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
matchResult <- distBalMatch(df=lalonde, treatCol= treatVal,
myBalCol = myBalVal, rhoExclude=r1s, rhoBalance=r2s, distList = pairDistVal,
propensityCols = psCols, maxIter=0)
```

`dummy` *This is a modified version of the function "dummy" from the R package dummies. Original code Copyright (c) 2011 Decision Patterns.*

Description

Change is made to the "model.matrix" function so that the output could be used for the current package.

Usage

```
dummy(
  x,
  data = NULL,
  sep = "",
  drop = TRUE,
  fun = as.integer,
  verbose = FALSE,
  name = NULL
)
```

Arguments

`x`	a data.frame, matrix or single variable or variable name
`data`	(optional) if provided, x is the name of a column on the data
`sep`	(optional) the separator used between variable name and the value
`drop`	(optional) whether to drop unused levels
`fun`	(optional) function to coerce the value in the final matrix; 'as,integer' by default
`verbose`	(optional) whether to print the number of variables; FALSE by default
`name`	(optional) the column name to be selected for converting; NULL by default

`edgelist2ISM` *Change the edgelist to the infinity sparse matrix*

Description

Change the edgelist to the infinity sparse matrix

Usage

```
edgelist2ISM(elist)
```

Arguments

`elist`	the vector of the edges

Value

the infinity sparse matrix object

`excludeCosts` *Create a skeleton representation of the exclusion edge costs*

Description

Create a skeleton representation of the exclusion edge costs associated with pairings for a given distance and network

Usage

```
excludeCosts(net, exclude.penalty = 1)
```

Arguments

`net`	the network structure
`exclude.penalty`	(optional) numeric penalty for excluding a treated unit; 1 by default

Value

the skeleton with exclusion edge cost

`extractEdges` *Extract edges from the network*

Description

Extract edges from the network

Usage

```
extractEdges(net)
```

Arguments

`net`	the network representation

Value

the list of edges

`extractSupply` *Extract the supply nodes from the net*

Description

Extract the supply nodes from the net

Usage

```
extractSupply(net)
```

Arguments

`net`	the network representation

Value

the vector of the supply nodes

`flattenSkeleton` *Turns a skeleton representation of edge costs in a network*

Description

Turns a skeleton representation of edge costs in a network back into the vector representation expected by the optimization routine. See comment on the costSkeleton function for more details.

Usage

```
flattenSkeleton(skeleton)
```

Arguments

`skeleton`	the skeleton structure

Value

vector representation expected by the optimization routine.

generateBalanceTable *Generate balance table*

Description

The helper function can generate tabular analytics that quantify covariate imbalance after matching. It only works for the 'Basic' version of matching (produced by `distBalMatch`).

Usage

```
generateBalanceTable(
  matchingResult,
  covList = NULL,
  display.all = TRUE,
  statList = c("mean.diffs")
)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch
`covList`	(optional) a vector of names of covariates used for evaluating covariate imbalance; NULL by default.
`display.all`	(optional) a boolean value indicating whether or not to show the data for all possible matches; TRUE by default
`statList`	(optional) a vector of statistics that are calculated for evaluating the covariate imbalance between treated and control group. The types that are supported can be found here: bal.tab.

Details

The result can be either directly used by indexing into the list, or post-processing by the function `compareTables` that summarizes the covariate balance information in a tidier table. Users can specify the arguments as follows: * `covList`: if it is set of NULL, all the covariates are included for the covariate balance table; otherwise, only the specified covariates will be included in the tabular result. * `display.all`: by default, the summary statistics for each match are included when the argument is set to TRUE. If the user only wants to see the summary statistics for matches with varying performance on three different objective values, the function would only display the matches with number of treated units being excluded at different quantiles. User can switch to the brief version by setting the parameter to FALSE. * `statList` is the list of statistics used for measuring balance. The argument is the same as `stats` argument in bal.tab, which is the function that is used for calculating the statistics. By default, only standardized difference in means is calculated.

Value

a named list object containing covariate balance table and statistics for numer of units being matched for each match; the names are the character of index for each match in the `matchResult`.

See Also

Other numerical analysis helper functions: generateRhoObj(), getUnmatched()

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Generate summary table for balance
balanceTables <- generateBalanceTable(matchResult)
balanceTableMatch10 <- balanceTables$'10'
```

generatePairdistanceBalanceGraph

Total variation imbalance vs. marginal imbalance

Description

Plotting function that generate sum of pairwise distance vs. total variation imbalance on specified balance variable. This function only works for 'Basic' version of matching (conducted using distBalMatch).

Usage

```
generatePairdistanceBalanceGraph(matchingResult)
```

Arguments

matchingResult	an object returned by the main matching function distBalMatch

Value

No return value, called for visualization of match result

See Also

Other Graphical helper functions for analysis: `generatePairdistanceGraph()`, `generateTVGraph()`

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Visualize the tradeoff between the pair-wise distance sum and
## total variation distance
generatePairdistanceBalanceGraph(matchResult)
```

`generatePairdistanceGraph` *Distance vs. exclusion*

Description

Plotting function that generate sum of pair-wise distance vs. number of unmatched treated units

Usage

```
generatePairdistanceGraph(matchingResult)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch

Value

No return value, called for visualization of match result

See Also

Other Graphical helper functions for analysis: `generatePairdistanceBalanceGraph()`, `generateTVGraph()`

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Generate visualization of tradeoff between pari-wise distance sum and
## number of treated units left unmatched
generatePairdistanceGraph(matchResult)
```

`generateRhoObj` *Penalty and objective values summary*

Description

Helper function to generate a dataframe with matching number, penalty (rho) values, and objective function values.

Usage

```
generateRhoObj(matchingResult)
```

Arguments

`matchingResult`	matchingResult object that contains information for all matches.

Value

a dataframe that contains objective function values and rho values corresponding coefficients before each objective function.

See Also

Other numerical analysis helper functions: `generateBalanceTable()`, `getUnmatched()`

Examples

```
## Generate matches
x1 = rnorm(100, 0, 0.5)
x2 = rnorm(100, 0, 0.1)
x3 = rnorm(100, 0, 1)
x4 = rnorm(100, x1, 0.1)
r1ss <- seq(0.1,50, 10)
r2ss <- seq(0.1,50, 10)
x = cbind(x1, x2, x3,x4)
z = sample(c(rep(1, 50), rep(0, 50)))
e1 = rnorm(100, 0, 1.5)
e0 = rnorm(100, 0, 1.5)
y1impute = x1^2 + 0.6*x2^2 + 1 + e1
y0impute = x1^2 + 0.6*x2^2 + e0
treat = (z==1)
y = ifelse(treat, y1impute, y0impute)
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
d1 <- as.matrix(dist(df["x1"]))
d2 <- as.matrix(dist(df["x2"]))
idx <- 1:length(z)
treatedUnits <- idx[z==1]
controlUnits <- idx[z==0]
d1 <- as.matrix(d1[treatedUnits, controlUnits])
d2 <- as.matrix(d2[treatedUnits, controlUnits])
matchResult1 <- twoDistMatch(df, "z", "y", dMat1=d1, dType1= "User", dMat2=d2,
dType2="User", myBalCol=c("x5"), rhoExclude=r1ss, rhoDistance=r2ss,
propensityCols = c("x1"))

## Generate tabular summary
generateRhoObj(matchResult1)
```

`generateTVGraph` *Marginal imbalance vs. exclusion*

Description

Plotting function that visualizes the tradeoff between the total variation imbalance on a specified variable and the number of unmatched treated units. This function only works for the 'Basic' version of matching (conducted using `distBalMatch`).

Usage

```
generateTVGraph(matchingResult)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch

Value

No return value, called for visualization of match result

See Also

Other Graphical helper functions for analysis: `generatePairdistanceBalanceGraph()`, `generatePairdistanceGraph()`

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree")
exactVal <- c("educ")
myBalVal <- c("race")
r1s <- c( 0.1, 0.3, 0.5, 0.7, 0.9,1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)
r2s <- c(0.01)
matchResult <- distBalMatch(df=lalonde, treatCol=treatVal, myBalCol=myBalVal,
rhoExclude =r1s, rhoBalance=r2s,
distList=pairDistVal, exactlist=exactVal,
propensityCols = psCols,ignore = c(responseVal), maxUnMatched = 0.1,
caliperOption=NULL, toleranceOption=1e-1, maxIter=0, rho.max.f = 10)

## Generate visualization of tradeoff between total variation distance and
## number of treated units left unmatched
generateTVGraph(matchResult)
```

`generate_rhos` *Generate rho pairs*

Description

An internal helper function that generates the set of rho value pairs used for matching. This function is used when exploring the Pareto optimality of the solutions to the multi-objective optimization in matching.

Usage

```
generate_rhos(rho1.list, rho2.list)
```

Arguments

`rho1.list`	a vector of rho 1 values
`rho2.list`	a vector of rho 2 values

Value

a vector of (rho1, rho2) pairs

`getExactOn` *Generate a factor for exact matching.*

Description

Generate a factor for exact matching.

Usage

```
getExactOn(dat, exactList)
```

Arguments

`dat`	dataframe containing all the variables in exactList
`exactList`	factor of names of the variables on which we want exact or close matching.

Value

factor on which to match exactly, with labels given by concatenating labels for input variables.

`getPropensityScore` *Fit propensity scores using logistic regression.*

Description

Fit propensity scores using logistic regression.

Usage

```
getPropensityScore(df, covs)
```

Arguments

`df`	dataframe that contains a column named "treat", the treatment vector, and columns of covariates specified.
`covs`	factor of column names of covariates used for fitting a propensity score model.

Value

vector of estimated propensity scores (on the probability scale).

`getUnmatched` *Get unmatched percentage*

Description

A function that generate the percentage of unmatched units for each match.

Usage

```
getUnmatched(matchingResult)
```

Arguments

`matchingResult`	matchingResult object that contains information for all matches

Value

data frame with three columns, one containing the matching index, one containing the number of matched units, and one conatining the percentage of matched units (out of original treated group size).

See Also

Other numerical analysis helper functions: `generateBalanceTable()`, `generateRhoObj()`

Examples

```
## Not run:
getUnmatched(matchResult)

## End(Not run)
```

`get_five_index` *An internal helper function that gives the index of matching with a wide range of number of treated units left unmatched*

Description

An internal helper function that gives the index of matching with a wide range of number of treated units left unmatched

Usage

```
get_five_index(matchingResult)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch

Value

a vector of five matching indices with the number of treated units excluded at 0th, 25th, 50th, 75th and 100th percentiles respectively.

---

makeInfinitySparseMatrix

Internal helper to build infinity sparse matrix

Description

Formats the data and make a call to InfinitySparseMatrix-class

Usage

```
makeInfinitySparseMatrix(
  data,
  cols,
  rows,
  colnames = NULL,
  rownames = NULL,
  dimension = NULL,
  call = NULL
)
```

Arguments

`data`	the input numeric vector of cost
`cols`	the input numeric vector corresponding to control units
`rows`	the input numeric vector corresponding to treated units
`colnames`	(optional) vector containing names for all control units
`rownames`	(optional) vector containing names for all treated units
`dimension`	(optional) vector of number of treated and control units
`call`	(optional) funtion call used to create the InfinitySparseMatrix

Value

an InfinitySparseMatrix object

`makeSparse` *Helper function to mask edges*

Description

Remove some of the treatment-control edges from a network flow representation of a match (forbidding those pairings)

Usage

```
makeSparse(net, mask, replaceMask = TRUE)
```

Arguments

`net`	the network object
`mask`	a matrix indicating whether to exclude the corresponding edge
`replaceMask`	(optional) whether to mask

Value

the masked network structure object

`matchedData` *Get matched dataframe*

Description

A function that returns the dataframe that contains only matched pairs from the original data frame with specified match index

Usage

```
matchedData(matchingResult, match_num)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch
`match_num`	Integer index of match that the user want to extract paired observations from

Value

dataframe that contains only matched pair data

Examples

```
## Generate Matches
x1 = rnorm(100, 0, 0.5)
x2 = rnorm(100, 0, 0.1)
x3 = rnorm(100, 0, 1)
x4 = rnorm(100, x1, 0.1)
r1ss <- seq(0.1,50, 10)
r2ss <- seq(0.1,50, 10)
x = cbind(x1, x2, x3,x4)
z = sample(c(rep(1, 50), rep(0, 50)))
e1 = rnorm(100, 0, 1.5)
e0 = rnorm(100, 0, 1.5)
y1impute = x1^2 + 0.6*x2^2 + 1 + e1
y0impute = x1^2 + 0.6*x2^2 + e0
treat = (z==1)
y = ifelse(treat, y1impute, y0impute)
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
d1 <- as.matrix(dist(df["x1"]))
d2 <- as.matrix(dist(df["x2"]))
idx <- 1:length(z)
treatedUnits <- idx[z==1]
controlUnits <- idx[z==0]
d1 <- as.matrix(d1[treatedUnits, controlUnits])
d2 <- as.matrix(d2[treatedUnits, controlUnits])
matchResult1 <- twoDistMatch(df, "z", "y", dMat1=d1, dType1= "User", dMat2=d2,
dType2="User", myBalCol=c("x5"), rhoExclude=r1ss, rhoDistance=r2ss,
propensityCols = c("x1"))
matchedData(matchResult1, 1)
```

`matched_index` *An internal helper function that translate the matching index in the sorted data frame to the original dataframe's row index*

Description

An internal helper function that translate the matching index in the sorted data frame to the original dataframe's row index

Usage

```
matched_index(matchingResult)
```

Arguments

`matchingResult`	an object returned by the main matching function distBalMatch

`matrix2cost` *change the distance matrix to cost*

Description

change the distance matrix to cost

Usage

```
matrix2cost(net, distance)
```

Arguments

`net`	the network structure
`distance`	distance matrix

Value

the vector of cost

`matrix2edgelist` *Helper function to convert matrix to list*

Description

Convert between a matrix representation of distances between treated and control units and a list of vectors (default format for build.dist.struct function in rcbalance package)

Usage

```
matrix2edgelist(mat)
```

Arguments

`mat`	matrix representation of distances between treated and control units

Value

list of vector representation of distances

`meldMask` *Helper function to combine two sparse distances*

Description

Combine two sparse distances, allowing only pairings allowed by both

Usage

```
meldMask(mask1, mask2)
```

Arguments

`mask1`	matrix of the first mask
`mask2`	matrix of the second mask

Value

combined mask structure

`netFlowMatch` *Create network flow structure*

Description

Create network flow structure

Usage

```
netFlowMatch(z, IDs = NULL)
```

Arguments

`z`	a vector of treatment vectors
`IDs`	(optional) the name of the units

Value

a networks structure

`obj.to.match` *An internal helper function that transforms the output from the RELAX algorithm to a data structure that is more interpretable for the output of the main matching function*

Description

An internal helper function that transforms the output from the RELAX algorithm to a data structure that is more interpretable for the output of the main matching function

Usage

```
obj.to.match(out.elem, already.done = NULL, prev.obj = NULL)
```

Arguments

`out.elem`	a named list whose elements are: (1) the net structure (2) the edge weights of pair-wise distance (3) the edge weights of marginal balance (4) the list of rho value pairs (5) the named list of solutions from the RELAX algorithm
`already.done`	a factor indicating the index of matches already been transformed
`prev.obj`	an object of previously transformed matches

Value

a named list with elements containing matching information useful for the main matching function

`pairCosts` *Create a skeleton representation of the edge costs*

Description

Create a skeleton representation of the edge costs

Usage

```
pairCosts(dist.struct, net)
```

Arguments

`dist.struct`	the distance structure
`net`	the net structure

Value

the skeleton representation of the given distance pairs and the net

rho_proposition *Generate penalty coefficient pairs*

Description

An internal helper function used for automatically generating the set of rho values used for grid search in exploring the Pareto optimal set of solutions.

Usage

```
rho_proposition(
  paircosts.list,
  rho.max.factor = 10,
  rho1old,
  rho2old,
  rho.min = 0.01
)
```

Arguments

`paircosts.list`	a vector of pair-wise distance.
`rho.max.factor`	a numeric value indicating the maximal rho values.
`rho1old`	a vector of numeric values of rho1 used before.
`rho2old`	a vector of numeric values of rho2 used before.
`rho.min`	smallest rho value to consider.

Value

a vector of pairs of rho values for future search.

solveP *Solve the network flow problem - basic version*

Description

Solve the network flow problem - basic version

Usage

```
solveP(net, f1.list, f2.list, rho, tol = 1e-05)
```

Arguments

`net`	the network representation
`f1.list`	the list of the first objective functions values for each node
`f2.list`	the list of the second objective functions values for each node
`rho`	the penalty coefficient
`tol`	the tolerance value for precision

Value

the solution represented in a named list

`solveP1` *Solve the network flow problem - twoDistMatch*

Description

Solve the network flow problem - twoDistMatch

Usage

```
solveP1(net, f1.list, f2.list, f3.list, rho1, rho2 = 0, tol = 1e-05)
```

Arguments

`net`	the network representation
`f1.list`	the list of the first objective functions values for each node
`f2.list`	the list of the second objective functions values for each node
`f3.list`	the list of the third objective functions values for each node
`rho1`	the penalty coefficient for the second objective
`rho2`	the penalty coefficient for the third objective
`tol`	the tolerance value for precision

Value

the solution represented in a named list

twoDistMatch — *Optimal tradeoffs among two distances and exclusion*

Description

Explores tradeoffs among three objective functions in multivariate matching: sums of two different user-specified covariate distances within matched pairs, and the number of treated units included in the match.

Usage

```
twoDistMatch(
  dType1 = "user",
  dType2 = "user",
  dMat1 = NULL,
  df = NULL,
  dMat2 = NULL,
  treatCol = NULL,
  distList1 = NULL,
  distList2 = NULL,
  rhoExclude = c(1),
  rhoDistance = c(1, 2, 3),
  myBalCol = NULL,
  exactlist = NULL,
  propensityCols = NULL,
  pScores = NULL,
  ignore = NULL,
  maxUnMatched = 0.25,
  caliperOption = NULL,
  toleranceOption = 0.01,
  maxIter = 0,
  rho.max.f = 10
)
```

Arguments

`dType1`	One of ("Euclidean", "Mahalanobis", "user") indicating the type of distance that are used for the first distance objective functions. NULL by default.
`dType2`	One of ("Euclidean", "Mahalanobis", "user") charactor indicating the type of distance that are used for the second distance objective functions. NULL by default.
`dMat1`	(optional) matrix object that represents the distance matrix using the first distance measure; `dType` must be passed in as "user" if dMat is non-empty
`df`	(optional) data frame that contain columns indicating treatment, outcome and covariates

`dMat2`	(optional) matrix object that represents the distance matrix using the second distance measure; `dType1` must be passed in as "user" if dMat is non-empty
`treatCol`	(optional) character, name of the column indicating treatment assignment.
`distList1`	(optional) character vector names of the variables used for calculating covariate distance using first distance measure specified by dType
`distList2`	(optional) character vector, names of the variables used for calculating covariate distance using second distance measure specified by dType1
`rhoExclude`	(optional) numeric vector, penalty values associated with the distance specified by `dMat` or `dType`. Default value is c(1).
`rhoDistance`	(optional) numeric vector, penalty values associated with the distance specified by `dMat1` or `dType1`. Default value is c(1,2,3).
`myBalCol`	(optional) character, column name of the variable on which to evaluate balance.
`exactlist`	(optional) character vector, names of the variables on which to match exactly; NULL by default.
`propensityCols`	character vector, names of columns on which to fit a propensity score model.
`pScores`	(optional) character, name of the column containing fitted propensity scores; default is NULL.
`ignore`	(optional) character vector of variable names that should be ignored when constructing the internal matching. NULL by default.
`maxUnMatched`	(optional) numeric, maximum proportion of unmatched units that can be accepted; default is 0.25.
`caliperOption`	(optional) numeric, the propensity score caliper value in standard deviations of the estimated propensity scores; default is NULL, which is no caliper.
`toleranceOption`	(optional) numeric, tolerance of close match distance; default is 1e-2.
`maxIter`	(optional) integer, maximum number of iterations to use in searching for penalty combintions that improve the matching; default is 0.
`rho.max.f`	(optional) numeric, the scaling factor used in proposal for rhos; default is 10.

Details

Matched designs generated by this function are Pareto optimal for the three objective functions. The degree of relative emphasis among the three objectives in any specific solution is controlled by the penalties, denoted by Greek letter rho. Larger values for the penalties associated with the two distances correspond to increased emphasis close matching on these distances, at the possible cost of excluding more treated units. Additional details:

- Users may either specify their own distance matrices (specifying the `user` option in `dType1` and/or `dType2` and supplying arguments to `dMat1` and/or `dMat2` respectively) or ask the function to create Mahalanobis or Euclidean distances on sets of covariates specified by the `distList1` and `distList2` arguments. If `dType1` or `dType2` is not specified, if one of these is set to `user` and the corresponding `dMat1` argument is not provided, or if one is NOT set to `user` and the corresponding `distList1` argument is not provided, an error will result.
- User-specified distance matrices passed to `dMat1` or `dMat2` should have row count equal to the number of treated units and column count equal to the number of controls.

- If the `caliperOption` argument is specified, a propensity score caliper will be imposed, forbidding matches between units more than a fixed distance apart on the propensity score. The caliper will be based either on a user-fit propensity score, identified in the input dataframe by argument `pScores`, or by an internally-fit propensity score based on logistic regression against the variables named in `psoreCols`. If `caliperOption` is non-NULL and neither of the other arguments is specified an error will result.
- `toleranceOption` controls the precision at which the objective functions is evaluated. When matching problems are especially large or complex it may be necessary to increase toleranceOption in order to prevent integer overflows in the underlying network flow solver; generally this will be suggested in appropariate warning messages.
- While by default tradeoffs are only assessed at penalty combinations provided by the user, the user may ask for the algorithm to search over additional penalty values in order to identify additional Pareto optimal solutions. `rho.max.f` is a multiplier applied to initial penalty values to discover new solutions, and setting it larger leads to wider exploration; similarly, `maxIter` controls how long the exploration routine runs, with larger values leading to more exploration.

Value

a named list whose elements are:

- "rhoList": list of penalty combinations for each match
- "matchList": list of matches indexed by number
- "treatmentCol": character of treatment variable
- "covs":character vector of names of the variables used for calculating within-pair distance
- "exactCovs": character vector of names of variables that we want exact or close match on
- "idMapping": numeric vector of row indices for each observation in the sorted data frame for internal use
- "stats": data frame of important statistics (total variation distance) for variable on which marginal balance is measured
- "b.var": character, name of variable on which marginal balance is measured (left NULL since no balance constraint is imposed here).
- "dataTable": data frame sorted by treatment value
- "t": a treatment vector
- "df": the original dataframe input by the user
- "pair_cost1": list of pair-wise distance sum using the first distance measure
- "pair_cost2": list of pair-wise distance sum using the second distance measure
- "version": (for internal use) the version of the matching function called; "Basic" indicates the matching comes from distBalMatch and "Advanced" from twoDistMatch.
- "fDist1": a vector of values for the first objective function; it corresponds to the pair-wise distance sum according to the first distance measure.
- "fExclude": a vector of values for the second objective function; it corresponds to the number of treated units being unmatched.
- "fDist2": a vector of values for the third objective function; it corresponds to the pair-wise distance sum corresponds to the

See Also

Other main matching function: `distBalMatch()`

Examples

```
x1 = rnorm(100, 0, 0.5)
x2 = rnorm(100, 0, 0.1)
x3 = rnorm(100, 0, 1)
x4 = rnorm(100, x1, 0.1)
r1ss <- seq(0.1,50, 10)
r2ss <- seq(0.1,50, 10)
x = cbind(x1, x2, x3,x4)
z = sample(c(rep(1, 50), rep(0, 50)))
e1 = rnorm(100, 0, 1.5)
e0 = rnorm(100, 0, 1.5)
y1impute = x1^2 + 0.6*x2^2 + 1 + e1
y0impute = x1^2 + 0.6*x2^2 + e0
treat = (z==1)
y = ifelse(treat, y1impute, y0impute)
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
d1 <- as.matrix(dist(df["x1"]))
d2 <- as.matrix(dist(df["x2"]))
idx <- 1:length(z)
treatedUnits <- idx[z==1]
controlUnits <- idx[z==0]
d1 <- as.matrix(d1[treatedUnits, controlUnits])
d2 <- as.matrix(d2[treatedUnits, controlUnits])
matchResult1 <- twoDistMatch(df, "z", "y", dMat1=d1, dType1= "User", dMat2=d2,
dType2="User", myBalCol=c("x5"), rhoExclude=r1ss, rhoDistance=r2ss,
propensityCols = c("x1"))
```

`visualize` *Visualize tradeoffs*

Description

Main visualization functions for showing the tradeoffs between two of the three objective functions.

Usage

```
visualize(
  matchingResult,
  x_axis = "dist1",
  y_axis = "dist2",
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  display_all = FALSE,
  cond = NULL,
  xlim = NULL,
  ylim = NULL,
  display_index = TRUE,
  average_cost = FALSE
)
```

Arguments

`matchingResult`	the matching result returned by either distBalMatch or twoDistMatch.
`x_axis`	character, naming the objective function shown on x-axis; one of ("pair", "marginal", "dist1", "dist2", "exclude"), "dist1" by default.
`y_axis`	character, naming the objective function shown on y-axis; one of ("pair", "marginal", "dist1", "dist2", "exclude"), "dist2" by default.
`xlab`	(optional) the axis label for x-axis; NULL by default.
`ylab`	(optional) the axis label for y-axis; NULL by default.
`main`	(optional) the title of the graph; NULL by default.
`display_all`	(optional) whether to show all the labels for match index; FALSE by default, which indicates the visualization function only labels matches at quantiles of number of treated units being excluded.
`cond`	(optional) NULL by default, which denotes all the matches are shown; otherwise, takes a list of boolean values indicating whether to include each match
`xlim`	(optional) NULL by default; function automatically takes the max of the first objective function values being plotted on x-axis; if specified otherwise, pass in the numeric vector c(lower_bound, upper_bound)
`ylim`	(optional) NULL by default; function automatically takes the max of the first objective function values being plotted on y-axis; if specified otherwise, pass in the numeric vector c(lower_bound, upper_bound)
`display_index`	(optional) TRUE by default; whether to display match index
`average_cost`	(optional) FALSE by default; whether to show mean cost

Details

By default, the plotting function will show the tradeoff between the first distance objective function and the marginal balance (if distBalMatch) is used; or simply the second distance objective function, if twoDistMatch is used.

Value

No return value, called for visualization of match result

Examples

```
## Generate matches
data("lalonde", package="cobalt")
psCols <- c("age", "educ", "married", "nodegree", "race")
treatVal <- "treat"
responseVal <- "re78"
pairDistVal <- c("age", "married", "educ", "nodegree", "race")
myBalVal <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
matchResult <- distBalMatch(df=lalonde, treatCol= treatVal,
myBalCol = myBalVal, rhoExclude=r1s, rhoBalance=r2s, distList = pairDistVal,
propensityCols = psCols, maxIter=0)
## Visualization
visualize(matchResult, "marginal", "exclude")
visualize(matchResult, "pair", "exclude")
```

Index