Package ‘rcbalance’

November 12, 2017

Type Package

Title Large, Sparse Optimal Matching with Refined Covariate Balance

Version 1.8.5

Date 2017-11-09

Author Samuel D. Pimentel

Maintainer Samuel D. Pimentel <spi@berkeley.edu>

Description Tools for large, sparse optimal matching of treated units and control units in observational studies. Provisions are made for refined covariate balance constraints, which include fine and near-fine balance as special cases. Matches are optimal in the sense that they are computed as solutions to network optimization problems rather than greedy algorithms.

Depends R (>= 3.2.0), MASS, plyr

License MIT + file LICENSE

Suggests optmatch, testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2017-11-12 17:28:28 UTC

R topics documented:

rcbalance-package *Sparse Optimal Matching with Refined Covariate Balance*

Description

This package computes sparse matches that are optimal under a set of refined covariate balance constraints. These constraints, provided by the user, are a set of nested categorical variables of decreasing imporance which must be marginally balanced as closely as possible in the resulting treated and matched control populations. For more detail see the references.

Details

Package:	rcbalance
Type:	Package
Version:	1.8.1
Date:	2016-02-10
License:	MIT + file LICENSE

The main function is `rcbalance`, which takes a distance/sparsity object containing information about matchability of the treated and control units and a list of fine balance variables and produces a match. The `build.dist.struct` function can be used to construct the distance/sparsity object from covariate information. The `count.pairings` function can be used to assess the sparsity of a proposed match. The other functions are largely for internal use and should not be needed by the large majority of users.

IMPORTANT NOTE: the functionality of this package is greatly enhanced if the `optmatch` package (v >= 0.9-1) is also loaded. In particular, when attempting to run the `rcbalance` command without having loaded `optmatch`, the users will receive an error message. The second reference below gives background on `optmatch`.

Author(s)

Samuel D. Pimentel <spi@wharton.upenn.edu>

References

Hansen, B.B. and Klopfer, S.O. (2006) Optimal full matching and related designs via network flows, JCGS 15 609-627.

Pimentel, S.D., Kelz, R.R., Silber, J.H., and Rosenbaum, P.R. (2015) Large, sparse optimal matching with refined covariate balance in an observational study of the health outcomes produced by new surgeons, JASA 110 (510), 515-527.

Pimentel, S.D. (2016) Large, sparse optimal matching with R package rcbalance, Obs. Studies 2, 4-23.

`build.dist.struct` *Build Distance Structure for Matching with Refined Balance*

Description

This function computes rank-based Mahalanobis distances between treated and control units and returns an object suitable for use in the distance.structure argument of rcbalance.

Usage

```
build.dist.struct(z, X, exact = NULL, 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`.

`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.

`calip.option` 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.cov` see `calip.option`.

`caliper` gives the size of the caliper when the user specifies the `calip.option` argument as 'propensity' or 'calip.cov'.

`verbose` if `TRUE`, prints output describing specific adjustments made in creating the distance objects.

Details

If `X` is a data frame and contains any character variables they are converted to factors with a warning. If there are missing values in factor columns of `X`, they are treated as a new factor level. If there are missing values in numeric or logical columns of `X`, an indicator of missingness for that column is added to `X` and the missing values are imputed with the column mean. This follows the recommendations of Rosenbaum (*Design of Observational Studies* section 9.4, 2010).

Value

A distance.structure object, the form of which is described in the documentation for the `distance.structure` argument of `rcbalance`. Treated and control indices are numbered `1:nt` and `1:nc` respectively based on the order in which they appear in the `z` vector.

Author(s)

Samuel D. Pimentel

See Also

`rcbalance`

`callrelax` *Solve Network Flow Problem using RELAX-IV Algorithm*

Description

An rcbalance method not meant to be called directly by users. Solves network flow optimization problems by calling the RELAX-IV algorithm, as implemented in FORTRAN by Dimitri Bertsekas and Paul Tseng.

IMPORTANT NOTE 1: the RELAX-IV code is not contained in this R package due to software license issues. Users can only access it by loading the `optmatch` package (>= 0.9-1) and accepting its license. The reference below gives background on `optmatch`.

Usage

```
callrelax(net)
```

Arguments

`net` a network flow problem, formatted as a list with the following arguments (where the network contains nnode nodes, numbered 1 through nnode and narc arcs):

- startn: a vector of length narc containing the node numbers of the start nodes of each arc in the network.
- endn: a vector of length narc containing the node numbers of the end nodes of each arc in the network.
- ucap: a vector of length narc containing the (integer) upper capacity of each arc in the network.
- cost: a vector of length narc containing the (integer) cost of each arc in the network.
- b: a vector of length nnode containing the (integer) supply or demand of each node in the network. Supplies are given as positive numbers and demands as negative numbers.

Value

A list with the following elements:

`crash` an integer, equal to zero if the algorithm ran correctly and equal to 1 if it crashed.

`feasible` an integer, equal to zero if the problem is infeasible and equal to 1 if it is feasible.

`x` a vector equal in length to the number of arcs in argument problem net, giving in each coordinate the number of units of flow passing across the corresponding edge in the optimal network flow.

Author(s)

Samuel D. Pimentel

References

Hansen, B.B. and Klopfer, S.O. (2006) Optimal full matching and related designs via network flows, JCGS 15 609-627.

`count.pairings` *Count treatment-control pairings.*

Description

Given a treatment indicator and a potential blocking variable, counts the number of allowed treatment-control pairings in the whole match within blocks of the proposed variable.

Usage

```
count.pairings(z, exact)
```

Arguments

`z` a vector of treatment indicators. Must contain exactly 2 distinct values, one for treated and one for control.

`exact` a vector of categories of a potential blocking variable. Must be the same length as argument `z`.

Value

The number of within-block treatment-control edges contained in the sparse match with the proposed blocks.

Author(s)

Samuel D. Pimentel

dist2net — *Building and Manipulating Network Flow Problems*

Description

These are internal rcbalance methods not meant to be called directly by users. They are used to construct a network flow problem from the information about a matching problem that is passed to the `rcbalance` method.

Usage

```
dist2net(dist.struct, k, exclude.treated = FALSE, ncontrol = NULL)

dist2net.matrix(dist.struct, k, exclude.treated = FALSE)

add.layer(net.layers, new.layer)

penalty.update(net.layers, newtheta, newp = NA)

penalize.near.exact(net.layers, near.exact)
```

Arguments

`dist.struct` An object specifying the sparsity structure of the match. For the dist2net method it is a list of vectors, and for the dist2net.matrix method it is a matrix or InfinitySparseMatrix. See rcbalance documentation for more details.

`k` a nonnegative integer. The number of control units to which each treated unit will be matched.

`exclude.treated` if `TRUE`, then when there is no feasible match using all treated units, a minimal number of treated units may be dropped so that a match can be formed. Specifying this argument adds penalized edges to the network so that such a match can be computed. NOTE: this argument is incompatible with values of `k` greater than 1.

`ncontrol` the number of controls in the matching problem. If left `NULL` (the default value), the value will be intuited from the maximum control label in the sparsity object.

`net.layers` a layered network object of the type produced by the dist2net function.

`new.layer` a vector equal in length to the number of treated and control units in the matching problem. Each coordinate contains the value of a new fine balance variable for the corresponding unit.

`newtheta` optional argument giving a new value for the theta field of the net.layers object (see value section for description of this field).

`newp` optional argument giving a new value for the p field of the net.layers object (see value section for description of this field).

`near.exact` a vector equal in length to the number of treated and control units in the matching problem. Edges between units with different values of this variable will be penalized.

Details

`dist2net` and `dist2net.matrix` take the distance structure given to `rcbalance` encoding information about the matching problem and converts it into a network flow problem. `add.layer` adds network structure to handle an individual fine balance variable (it can be called iteratively to add many such variables). `penalty.update` is used to change the penalties for each layer (and the penalties for edges used to exclude treated units if they are present) and `penalize.near.exact` is used to add penalties to the treated-control edges to allow near-exact matching. See the references for a detailed description of how the matching problem is transformed into a network.

Value

A layered network object, formatted as a list with the following arguments (where narcs is the number of arcs and nnodes is the number of nodes in the network):

`startn` a vector of length narc containing the node numbers of the start nodes of each arc in the network.

`endn` a vector of length narc containing the node numbers of the end nodes of each arc in the network.

`ucap` a vector of length narc containing the (integer) upper capacity of each arc in the network.

`cost` a vector of length narc containing the (integer) cost of each arc in the network.

`b` a vector of length nnode containing the (integer) supply or demand of each node in the network. Supplies are given as positive numbers and demands as negative numbers.

`tcarcs` an integer giving the total number of arcs between the treated and control nodes in the network.

`layers` a list object containing information about the refined covariate balance layers of the network.

`z` a vector of treatment indicators.

`fb.structure` a matrix containing information about the membership of the treated and control units in the different classes of refined balance covariates.

`penalties` a vector of integer penalties, one for each fine balance layer.

`theta` a value no less than 1 giving the ratio by which the penalty is increased with each additional layer of fine balance.

`p` a nonnegative value giving the penalty for the finest level of fine balance.

Author(s)

Samuel D. Pimentel

`rcbalance` *Optimal Matching with Refined Covariate Balance*

Description

This function computes an optimal match with refined covariate balance.

Usage

```
rcbalance(distance.structure, near.exact = NULL, fb.list = NULL,
treated.info = NULL, control.info = NULL, exclude.treated = FALSE, target.group = NULL,
 k = 1, penalty = 3, tol = 1e-5)
```

Arguments

`distance.structure`

a list of vectors that encodes information about covariate distances between treated and control units. The list is equal in length to the number of treated units. Each vector corresponds to a treated unit and is equal in length to the number of control units to which it can be matched. It is assumed that there are a total of `nc` control units in the problem and that they are numbered from 1 to `nc`. The names of each vector in the list give the index (in the vector `1:nc`) of the control units to which the treated unit in question can be matched, and the elements of each vector are the covariate distances between the treated unit and the corresponding control. Note that for a dense matching problem (in which each treated unit can be matched to any control), every vector in the list will have length nc and rownames 1 through nc.

Alternatively, this same information can be passed as a `matrix` or `InfinitySparseMatrix` with rows corresponding to treated units and columns corresponding to controls. Entries given as `Inf` correspond to pairs that cannot be matched.

`near.exact` an optional character vector specifying names of covariates for near-exact matching. This argument takes precedence over any refined covariate balance constraints, so the match will produce the best refined covariate balance subject to matching exactly on this variable wherever possible. If multiple covariates are named, near-exact matching will be done on their interaction.

`fb.list` an optional list of character vectors specifying covariates to be used for refined balance. Each element of the list corresponds to a level of refined covariate balance, and the levels are assumed to be in decreasing order of priority. Each character vector should contain one or more names of categorical covariates on which the user would like to enforce near fine balance. If multiple covariates are specified, an interaction is created between the categories of the covariates and near fine balance is enforced on the interaction. IMPORTANT: covariates or interactions coming later in the list must be nested within covariates coming earlier in the list; if this is not the case the function will stop with an error. An easy way to ensure that this occurs is to include in each character vector all the variables named in earlier list elements. If the `fb.list` argument is specified, the `treated.info` and `control.info` arguments must also be specified.

`treated.info` an optional data frame containing covariate information for the treated units in the problem. The row count of this data frame must be equal to the length of the `distance.structure` argument, and it is assumed that row `i` contains covariate information for the treated unit described by element `i` of `distance.structure`. In addition, the column count and column names must be identical to those of the `control.info` argument, and the column names must include all of the covariate names mentioned in the `near.exact` and `fb.list` arguments.

`control.info` an optional data frame containing covariate information for the control units in the problem. The row count of this data frame must be no smaller than the maximum control index in the `distance.structure` argument, and it is assumed that row `i` contains the covariate information for the control indexed by `i` in distance.structure. In addition, the column count and column names must be identical to those of the `treated.info` argument.

`exclude.treated`

if `TRUE`, then when there is no feasible match using all treated units, a minimal number of treated units will be dropped so that a match can be formed. The excluded treated units will be selected optimally so that the cost of the matching is reduced as much as possible. NOTE: `exclude.treated` = `TRUE` is incompatible with arguments to `target.group` and with values of `k` larger than 1.

`target.group` an optional data frame of observations with the desired covariate distribution for the selected control group, if it differs from the covariate distribution of the treated units. This argument will be ignored unless `fb.list`, `treated.info` and `control.info` are also specified, and it must have the same dimensions as `treated.info`.

`k` a nonnegative integer. The number of control units to which each treated unit will be matched.

`penalty` a value greater than 1. This is a tuning parameter that helps ensure the different levels of refined covariate balance are prioritized correctly. Setting the penalty higher tends to improve the guarantee of match optimality up to a point, but penalties above a certain level cause integer overflows and throw errors. Usually it is not recommended that the user change this parameter from its default value.

`tol` edge cost tolerance. This is the smallest tolerated difference between matching costs; cost differences smaller than this will be considered zero. Match distances will be scaled by inverse tolerance, so when matching with large edge costs or penalties the tolerance may need to be increased.

Details

In order to perform matching, `rcbalance` requires the user to load the `optmatch` (>= 0.9-1) package separately. The manual loading is required due to software license issues. If the package is not loaded the `rcbalance` command will fail with an error saying the `optmatch` package is not present.

The second reference below gives background on `optmatch`.

Value

A list with the following components:

`matches` a nt by k matrix containing the matched sets produced by the algorithm (where nt is the number of treated units). The rownames of this matrix are the numbers of the treated units (indexed by their position in distance.structure), and the elements of each row contain the indices of the control units to which this treated unit has been matched.

`fb.tables` a list of matrices, equal in length to the fb.list argument. Each matrix is a contingency table giving the counts among treated units and matched controls for each level of the categorical variable specified by the corresponding element of fb.list.

Author(s)

Samuel D. Pimentel

References

Pimentel, S.D., Kelz, R.R., Silber, J.H., and Rosenbaum, P.R. (2015) Large, sparse optimal matching with refined covariate balance in an observational study of the health outcomes produced by new surgeons, JASA 110 (510), 515-527.

Hansen, B.B. and Klopfer, S.O. (2006) Optimal full matching and related designs via network flows, JCGS 15 609-627.

Examples

```
## Not run:
library(optmatch)
data(nuclearplants)

#require exact match on variables ne and pt, use rank-based Mahalanobis distance
my.dist.struct <- build.dist.struct(z = nuclearplants$pr,
X = subset(nuclearplants[c('date','t1','t2','cap','bw','cum.n')]),
exact = paste(nuclearplants$ne, nuclearplants$pt, sep = '.'))

#match with refined covariate balance, first on ct then on (ct x bw)
rcbalance(my.dist.struct, fb.list = list('ct',c('ct','bw')),
  treated.info = nuclearplants[which(nuclearplants$pr ==1),],
  control.info = nuclearplants[which(nuclearplants$pr == 0),])

#repeat the same match using match_on tool from optmatch and regular Mahalanobis distance
exact.mask <- exactMatch(pr ~ ne + pt, data = nuclearplants)
my.dist.matrix <- match_on(pr ~ date + t1 + t2 + cap + bw + cum.n,
within = exact.mask, data = nuclearplants)
match.matrix <-
rcbalance(my.dist.matrix*100, fb.list = list('ct',c('ct','bw')),
treated.info = nuclearplants[which(nuclearplants$pr ==1),],
control.info = nuclearplants[which(nuclearplants$pr == 0),])

## End(Not run)
```

Index